McCarl GAMS User Guide

onomicsTexas A&M UniversityAlex MeerausPaul van der EijkMichael BussieckSteven DirkseFranz

Table of Contents

Foreword 0

Chapter I McCarl GAMS User Guide 1

1 Forword 2

2 Introduction 3

Chapter II Quick Start Tutorial 4

1 Basic models 4

Solving an optimization problem 5

Solving for an economic equilibrium 5

Solving a nonlinear equation system 6

2 Dissecting the simple models 7

Variables 7

What is the new Z variable in the optimization problem? 8

Equations 9

.. specifications 10

Model 11

Solve 12

Why does my nonlinear equation system maximize something? 13

What are the .L items 14

3 Running the job 14

Command line approach 14

IDE approach 15

4 Examining the output 15

Echo print 15

Incidence of compilation errors 16

Symbol list and cross reference maps 17

Generation listing 18

Equation listing 18

Variable listing 19

Model statistics 21

Execution output 21

Solver report 21

Solution summary 21

Equation solution report 22

Variable solution report 23

5 Exploiting algebra 25

Equation writing – sums 25

Revised algebra exploiting optimization example 25

Revised equilibrium example 27

6 Dissecting the algebraic model 28

Sets 28

Alias 29

Data entry 29

Scalars 30

Parameters 30

© 2022 Prof. Bruce McCarl

Tables ... 31
Direct assignment ... 32
Algebraic nature of variable and equation specifications ... 32
Algebra and model .. specifications ... 33
Output differences ... 34
Equation listing ... 34
Variable list ... 35
Equation solution report ... 36
Variable solution report ... 36
7 Good modeling practices ... 36
8 Structure of GAMS statements, programs and the ; ... 38
9 Adding complexity ... 39
Conditionals ... 39
Conditionally execute an assignment ... 39
Conditionally add a term in sum or other set operation ... 39
Conditionally define an equation ... 40
Conditionally include a term in an equation ... 40
Displaying data ... 40
Report writing ... 42
10 Why use GAMS and algebraic modeling ... 43
Use of algebraic modeling ... 43
Context changes ... 44
Expandability ... 44
Augmentation ... 45
Aid with initial formulation and subsequent changes ... 46
Adding report writing ... 47
Self-documenting nature ... 47
Large model facilities ... 47
Automated problem handling and portability ... 48
Model library and widespread professional use ... 48
Use by Others ... 49
Ease of use with NLP, MIP, CGE and other problem forms ... 49
Interface with other packages ... 49
Chapter III Language Basics 49
1 Sets ... 49
Set declaration ... 50
Singleton Sets ... 51
Subsets ... 53
Element definition ... 54
Explicit element definition ... 54
Set definition through Tables ... 55
Element definition by computation ... 56
Multi dimensional sets ... 56
Domain checking ... 57
Set element referencing ... 58
Whole sets ... 58
Single elements ... 59
Operating over part of a set ... 59
Using subsets ... 59
Using conditionals ... 60
Sameas and Diag ... 60

© 2022 Prof. Bruce McCarl

Ord and Card ... 61
Using tuples ... 61
Defining a tuple with the matching and # operators ... 62
Universal Set: * as a set identifier ... 63
Using set attributes ... 64
Finding sets from data ... 65
Using another name or an alias ... 66
Element order and capitalization in output ... 67
Functions specifically referencing sets ... 67
Ord ... 67
Ordered and Unordered sets ... 67
Card ... 68
Sameas ... 68
Diag ... 69
Indexing sets defined over time ... 69
Leads and Lags: + / - ... 69
Circular or Equilibrium Leads and Lags: ++ / -- ... 70
Element Position ... 70
Set Arithmetic ... 71
Unions ... 71
Intersections ... 71
Complements ... 71
Differences ... 71
2 Data Entry ... 72
Scalars ... 72
Parameters ... 73
Table ... 74
Calculated data ... 76
3 Variables, Equations, Models and Solves ... 77
Variables ... 77
Variable Declaration ... 77
Variable attributes ... 79
Assigning variable and equation attributes ... 81
Equation ... 82
Equation Declaration ... 82
.. Equation specifications ... 83
Equation attributes ... 87
Assigning equation attributes ... 87
Model ... 87
Model attributes ... 90
List of attributes ... 90
Solve: Maximizing, Minimizing, and Using ... 93
Actions on executing solve ... 95
Programs with multiple solve statements ... 96
Multiple solve management - merge replace ... 97
Choosing a solver ... 98
4 Model Types and Solvers ... 98
Model Types ... 99
Linear programs (LP) ... 99
Nonlinear program (NLP) ... 99
Quadratically constrained program (QCP) ... 100
Mixed integer programming (MIP) ... 100
Relaxed mixed integer programming (RMIP) ... 101

© 2022 Prof. Bruce McCarl

Mixed complementarity problem (MCP) ... 102
Mixed integer nonlinear program (MINLP) ... 103
Relaxed mixed integer nonlinear program (RMINLP) ... 104
Mixed integer quadratically constrained program (MIQCP) ... 104
Relaxed mixed integer quad. constrain program (RMIQCP) ... 105
Constrained nonlinear systems (CNS) ... 105
Mathematical program with equilibrium constraints (MPEC) ... 106
Nonlinear programming with discontinuous derivatives (DNLP) ... 107
Relaxed mathematical program with equilibrium constraints (RMPEC) ... 107
Extended Mathematical Programs (EMP) ... 107
Solver capabilities matrix ... 108
Solvers ... 108
General notes on solver licensing ... 108
General notes on solver versions ... 109
Available solvers ... 110
ALPHAECP ... 111
ANTIGONE ... 111
BARON ... 112
BONMIN/BONMINH ... 112
CBC ... 113
CSDP ... 114
CONOPT ... 114
CONOPTD ... 114
CONVERT ... 114
CONVERTD ... 115
CPLEX ... 115
CPLEXD ... 115
DE ... 115
DEA ... 116
DECIS/DECISC/DECISM ... 116
DICOPT ... 116
EMP ... 116
EMPSP ... 117
EXAMINER ... 117
GAMSCHK ... 117
GLOMIQO ... 118
GUROBI ... 118
GUSS ... 118
IPOPT/IPOPTH ... 126
JAMS ... 126
KESTREL ... 127
KNITRO ... 127
LGO ... 127
LINDO/LINDOGLOBAL ... 128
LOGMIP ... 128
MILES ... 128
MILESE ... 129
MILESOLD ... 129
MINOS ... 129
MINOS5 ... 129
MOSEK ... 129
MPSGE ... 129
MPS2GMS ... 130
MSNLP ... 130

© 2022 Prof. Bruce McCarl

NLPEC 130
OSICplex 130
OSIGurobi 130
OSIMosek 131
OSIXPRESS 131
OSISOPLEX 131
PATH/PATHNLP 131
PATHC 132
PATHNLP 132
PATHOLD 132
SBB 132
SCENRED 132
SCIP 132
SNOPT 133
SOPLEX 133
XA 133
XAPAR 133
XPRESS 133
Choosing a solver 134
5 Standard Output 134
Where is my output? LOG and LST files 134
Output overview and navigation 134
GAMS phases and output generated 137
Compilation phase output 137
Echo print of the input file 137
Compilation phase error messages 138
Repositioning error messages 141
Symbol reference map 142
Symbol listing 144
Unique element list 145
Unique element cross reference 146
Execution output 146
Display output 147
Execution error output 147
Symptoms of the presence of an execution error 147
Output produced by a solve statement 147
Model generation error listing 148
Equation listing 149
Variable listing 151
Model characteristics statistics 152
Model generation time 153
Solve summary 153
Common solver report 153
Solver report 154
The variable and equation solution listing 155
Including slacks in the output 156
Ranging analysis 156
Final execution summary 161
Report summary 161
File summary 161
Managing output pages 161
Page width and height 162
New pages 162
Adding an output title to each page 163

© 2022 Prof. Bruce McCarl

Managing output volume ... **163**
Eliminate model listing ... 163
Eliminate cross reference map ... 164
Eliminate symbol list ... 164
Eliminate solution output ... 164
Eliminate echo print ... 164
Restrict output just to a few displays ... 164
Adding slack variables to the output ... **164**
Sending messages to the LOG file ... **165**
6 Writing Models and Good Modeling Practices ... **165**
Formatting models - my recommendations ... **165**
Use longer names and descriptions ... 166
Basic point ... 167
Include comments on procedures and data nature and sources ... 168
Entering raw versus calculated data ... 169
Avoiding use of * in input data set specification ... 169
Making sets work for you ... 170
Making subsets work for you ... 170
Formatting the typing of files to improve model readability ... 171
Other possible conventions ... 173
Chapter IV Changing licenses **174**
1 Licenses on IDE ... **174**
2 Licenses outside of IDE—Windows and Unix/Linux ... **174**
Chapter V Running Jobs with GAMS and the GAMS IDE **175**
1 Basic approaches to GAMS usage ... **175**
2 Running GAMS from the command line ... **175**
3 IDE concept and usage ... **176**
Steps to using IDE ... **177**
Create a project ... 177
Defining a project name and location ... 177
Creating or opening an existing GMS file ... 178
Preparing file for execution ... 179
Select default IDE functions ... 180
Page size and LST file opening ... 180
Make IDE the default GMS file processor ... 181
Run GAMS by clicking the run button ... 181
Open and navigate around the output ... 183
Using the process window ... 184
Using the LST file navigation window ... 185
Finding the Active Location ... 188
Working with your own file ... **189**
Fixing compilation errors ... **190**
Selected techniques for use of the IDE ... **192**
Ways to find and/or replace text strings ... 192
Search menu and find in files ... 193
Matching parentheses ... 194
Moving column blocks ... 195
Altering syntax coloring ... 195
Finding out more through help ... **196**

© 2022 Prof. Bruce McCarl

Help on the IDE ... 197
Help on GAMS ... 198
Accessing help on solvers ... 200
Adding your own documentation ... 201
Accessing documentation outside the IDE ... 202
Unraveling complex files: Refreader ... 202
Basic output ... 202
Symbol Tab ... 203
Files used Tab ... 203
Sets, Parameters etc. Tabs ... 204
Unused Tab ... 205
Steps to Using Refreader ... 206
Saving the Refreader output ... 207
Differencing files ... 208
Spell checking in files ... 210
Saving and Using a Script ... 210
When is it not worth using? ... 212
Employing command line parameters ... 212
A difficulty you will have using IDE ... 213
Installation ... 213
Install GAMS and on Windows machines the IDE ... 213
On Windows machines make IDE icon ... 214
On Linux/Unix run Gamsinst ... 214
Choosing solvers ... 214
Solver choice outside of IDE ... 215
Unpacking software on Windows machines ... 216

Chapter VI Fixing Compilation Errors 216

1 Don't bark up the wrong tree ... 216
2 Finding errors: ** ... 216**
3 Finding errors: $ ... 217
4 Repositioning error messages: Errmsg ... 217
5 Improperly placed semi colons - error A ... 218
6 Error message proliferation ... 219
7 Commonly found errors and their cause ... 219
8 Other common errors ... 220
Excess or insufficient semi colons - error B ... 221
Spelling mistakes - error C ... 221
Omitted Set elements - error D ... 222
Indexing problems - error E ... 223
Summing over sets already indexed - error F ... 223
Neglecting to deal with sets - error G ... 224
Mismatched parentheses - error H ... 224
Improper equation ".." statements - error I ... 225
Entering improper nonlinear expressions - error J ... 226
Using undefined data - error K ... 227
Improper references to individual set elements - error L ... 228
No variable, parameter, or equation definition - error M ... 229
Duplicate names - error N ... 229
Referencing item with wrong set - error O ... 230
ORD on an unordered set - error P ... 230

© 2022 Prof. Bruce McCarl

Chapter VII More Language Basics 231

1 Rules for Item Names, Element Names and Explanatory Text 232

Item name rules 232
Element name rules 232
Explanatory text rules 233

2 Including Comments 234

Blank lines 235
Single line comments 235
Multiple line comments 236
End of line comments 236
In line comments 237
Outside margin comments 238
Hidden comments 239

3 Calculating Items 239

Basic components of calculations 239
Operators 240
= 240
.. statements 240
Basic arithmetic + - * / ** 241
Arithmetic hierarchy 241
Changing hierarchy using parentheses 242
Operations over set dependent items 242
= replacement or .. statements 242
Sum , Smax, Smin, Prod 242
Sum 242
Smin Smax 243
Prod 244
Alternative set addressing schemes 244
Avoiding set domain errors 244
Multiple sets 245
Conditionals to restrict set coverage 245
Tuples and subsets to restrict set coverage 246
Items that can be calculated 247
Sets 247
Data 247
Equation calculations 248
Acronyms 248
Cautions about calculations 249
Dynamic 249
Static 249
Repeated static 249
Cautions about dynamic /static calculations 249
Potential other components in calculations 251
Mixing logical expressions, sets and numbers 251
Functions 252
Intrinsic Functions 252
Common mathematical functions 252
Abs 253
Execseed 253
Exp 253
Ifthen 254
Log, Log10, Log2 254

© 2022 Prof. Bruce McCarl

Max , Min 254
Prod 254
Round 255
Smin , Smax 255
Sqr 255
Sqrt 255
Sum 256
Other Mathematical functions 256
Logical Functions 260
Time and Calender functions 261
GAMS utility and performance functions 263
String manipulation functions: Ord, Ordascii, Ordebcdic 266
String manipulation functions: Card 267
Computer Characteristic Functions 267
Extrinsic Functions 268
Probability Distribution Function use in models 271
Random Numbers from Functions 275
Defining an Extrinsic Function 282
Special values 283
Inf, -Inf 283
Na 283
Eps 283
Undf 283
Yes/No 284
Model and optimal solution items 284
Attributes of variables and equations 284
L 284
M 285
Lo 286
.range 286
Up 287
Fx 287
Scale 288
Prior 288
Attributes of models 289
Including conditionals 289
Right and left hand side conditionals 289
4 Improving Output via Report Writing 290
Adding report writing 291
Basics of solution based report writing calculations 291
Adding a report 292
Notes on indefinite sets in parameter statements 293
Using displays 293
Abort 294
Controlling displays 294
Formatting display decimals and layout 294
Taking control of display decimals 297
Controlling item ordering 299
Controlling item capitalization 299
Formatting pages and lines 299
Output via put commands 300
Reordering set order in output 300
Preprogrammed table making utility: Gams2tbl 301
Output to other programs 302

© 2022 Prof. Bruce McCarl

Obtaining graphical output ... **302**
Sorting output ... **303**
Sorting in GAMS ... 304
Rank ... 304
5 Rules for Item Capitalization and Ordering ... **306**
Item capitalization ... **306**
Reviewing capitalization: $Onsymlist and $Onuellist ... 306
Set element order ... **307**
Reviewing set element ordering: $Onuellist ... **310**
6 Conditionals ... **310**
Basic forms of conditionals ... **310**
$ conditionals ... 311
Ways $ conditionals are employed ... 311
Suppressing calculation of items (left hand side) ... 311
Suppressing terms in equations (right hand side) ... 312
Controlling indices in sums etc. ... 313
Suppressing model equations (left hand side) ... 314
Conditionally displaying information ... 315
Terminating a program: Abort ... 316
If, Else, and Elseif ... 316
While ... 317
Repeat ... 317
Conditional placement and program execution speed ... **317**
Forms of conditional / logical true false statements ... **317**
Numerical comparisons ... 317
Eq: = ... 318
Ne:<> ... 318
Gt: > ... 319
Lt: < ... 319
Ge: >= ... 319
Le: <= ... 320
Eqv: <=> Imp: -> ... 320
Data existence ... 321
Existence/nonzero data item or result ... 321
Computation over a set ... 322
Set comparisons ... 323
Element position: Ord and Card ... 323
Element text comparison: Sameas and Diag ... 323
Subset or tuple membership ... 324
Acronym comparisons ... 325
Nesting logical conditions ... **326**
Nesting operators ... 326
And ... 326
Or ... 327
Xor ... 327
Not ... 328
Nested $ conditionals ... 329
Nested Operators and precedence order ... 329
Note of caution ... 329
The conditional alternative: the tuple ... **330**
7 Control Structures ... **331**
If, Else, and Elseif ... **331**
Alternative syntax ... 334

© 2022 Prof. Bruce McCarl

Endif ... 334
Loop ... 334
Alternative syntax ... 336
Endloop ... 336
While ... 337
Alternative syntax ... 338
Endwhile ... 338
For, To, Downto, and By ... 339
Alternative syntax ... 340
Endfor ... 340
Repeat, Until ... 341

Chapter VIII Doing a Comparative Analysis with GAMS 342

1 Basic approaches ... 342
2 Manual approach ... 342
Introducing cross scenario report writing ... 345
Percentage change cross scenario reports ... 346
Preserving changed data ... 347
3 An automated approach - avoiding repeated work ... 348
Adding a scenario ... 351
Changing model structure ... 351
4 Where am I? ... 352

Chapter IX GAMS Command Line Parameters 353

1 Important parameters ... 354
Compiler function – regional settings ... 354
Error detection and correction ... 354
LST and LOG output content and format control ... 355
Solver name choice ... 356
Option file presence ... 357
Directory management ... 357
Saves and restarts ... 358
More Secure "obfuscated" saves and restarts ... 359
User defined options ... 361
2 Alphabetic list of all GAMS command line parameters ... 361
-- // -/ /-- on command lines ... 363
Action: A ... 363
Appendexpand: Ae ... 364
Appendlog: Al ... 364
Appendout: Ao ... 364
Bratio ... 364
Case ... 365
Cerr ... 365
Charset ... 365
CNS ... 366
Curdir: Cdir ... 366
Dformat: Df ... 366
DNLP ... 367
Domlim ... 367
Dumpopt ... 367
Dumpparms: Dp ... 370

© 2022 Prof. Bruce McCarl

DumpParmsLogPrefix: DPLP ... 371
EMP ... 372
Eolonly: Ey ... 372
Errmsg ... 372
Errnam ... 373
Error ... 373
Errorlog: Er ... 373
Etlim: Etl ... 373
Execerr ... 374
Execmode ... 374
Expand: Ef ... 374
FDDelta ... 375
FDOpt ... 376
Ferr ... 377
Filecase ... 377
FileStem ... 377
Forcework: Fw ... 378
Forlim ... 378
Fsave ... 378
G205 ... 378
Gdx ... 379
Gdxcompress ... 379
Gdxconvert ... 379
GdxUELs ... 380
Griddir: Gdir ... 380
Gridscript: Gscript ... 380
HeapLimit: Hl ... 381
Holdfixed ... 381
Ide ... 381
Input: I ... 382
Inputdir: Idir ... 382
Inputdir1 to inputdir40: Idir1 to idir40 ... 383
Integer1 to Integer5 ... 383
IntVarUp ... 383
iterlim ... 384
Jobtrace: Jt ... 384
Keep ... 385
Libincdir: Ldir ... 385
License ... 385
Limcol ... 385
Limrow ... 386
Logfile: Lf ... 386
Logline: Ll ... 386
Logoption: Lo ... 387
LP ... 387
MaxProcDir ... 387
MCP ... 387
McprHoldFx ... 388
MINLP ... 388
MIP ... 388
MIQCP ... 388
MPEC ... 388
Multipass: Mp ... 389
NLP ... 389

© 2022 Prof. Bruce McCarl

Nodlim ... 389
NoNewVarEqu ... 389
On115 ... 390
Optca ... 390
Optcr ... 390
Optdir ... 390
Optfile ... 391
Output: O ... 391
Pagecontr: Pc ... 391
Pagesize: Ps ... 391
Pagewidth: Pw ... 392
Parmfile: Pf ... 392
Plicense ... 393
ProcDir ... 393
Profile ... 393
Profilefile ... 394
Profiletol: Ptol ... 394
Putdir: Pdir ... 395
QCP ... 395
Reference: Rf ... 395
Reslim ... 395
Restart: R ... 396
RMINLP ... 396
RMIP ... 396
RMIQCP ... 396
RMPEC ... 397
Save: S ... 397
Savepoint: Sp ... 397
Scrdir: Sd ... 398
Scrext: Se ... 398
Scriptexit ... 398
Scriptfrst: Sf ... 398
Scriptnext: Script ... 399
Scrnam: Sn ... 399
Seed ... 399
Solprint ... 399
Solvelink: Sl ... 400
Solver ... 402
Solvercntr: Scntr ... 402
Solverdict: Sdict ... 402
Solverinst: Sinst ... 402
Solvermatr: Smatr ... 403
Solversolu: Ssolu ... 403
Solverstat: Sstat ... 403
Stepsum ... 403
strictSingleton ... 404
Stringchk ... 404
Subsys ... 405
Suppress ... 405
Symprefix ... 405
Symbol ... 405
Sys10 ... 405
Sys11 ... 406
Sysdir ... 406

© 2022 Prof. Bruce McCarl

Sysincdir: Sdir ... 406
Sysout ... 407
Tabin ... 407
Tformat: Tf ... 407
Threads ... 407
ThreadsAsync ... 408
Timer ... 409
Trace ... 409
Traceopt ... 409
User1 to user5: U1 to U5 ... 409
Warnings ... 410
Workdir: Wdir ... 410
Workfactor ... 410
Workspace ... 410
Xsave: Xs ... 411
Zerores ... 411
Zeroresrep ... 411

Chapter X Saves and Restarts 411

1 Save Restart Basics ... 412
Save: S ... 413
Restart: R ... 413
Xsave: Xs ... 414
2 Use of save and restarts and their effect ... 414
Save and restart on command line ... 414
IDE usage ... 415
Save and restart calling GAMS from within GAMS ... 415
3 Why use save and restart? ... 416
Increasing run efficiency ... 416
Output management ... 417
Separation of code function ... 417
Save study results ... 418
Comparative statics analysis ... 418
Compiled code ... 418
Fast related solutions ... 418

Chapter XI Customizing GAMS 419

1 What types of options are there? ... 419
2 Possible command line parameters to customize ... 419
3 How can these options be set? ... 420
To set in command line via pf= ... 420
To set in Gmsprmxx.txt parameter file ... 421
To set in IDE ... 422
4 Hierarchy for customizing options ... 423
5 List of all customizing options ... 423

Chapter XII Finding and Fixing Errors or Performance Problems 439

1 Fixing Execution Errors ... 439

© 2022 Prof. Bruce McCarl

GAMS limit errors **440**
Arithmetic errors during GAMS execution **441**
Execution errors during model generation **442**
Execution errors during model solution **444**
Solver function evaluation errors 445
Symptoms 445
Allowing errors to occur 446
Repair 446
Presolve errors 447
Problem eliminated 447
No feasible mixed integer solution 448
No feasible continuous solution 449
Solver specific limits 450
Basing conditionals on number of errors **450**
Clearing error conditions **451**
2 Scaling GAMS Models **451**
Basics **451**
Theory of scaling **452**
Scaling a variable 452
Scaling equations 453
Simultaneous equation and variable scaling **454**
Example of scaling 455
Scaling of GAMS models **456**
Scaling in GAMS solvers 456
Using GAMS scaling assistance **457**
Why should you scale? 458
Effect of scaling on GAMS output **458**
How do you know how much to scale? **459**
A caution when scaling – runaway cases **459**
User defined data scaling **460**
Nonlinear scaling **460**
3 Small to Large: Aid in Development and Debugging **460**
Basics **461**
Expandability in an example 461
Essence of the small to large approach **463**
Steps for working from small to large. **464**
Making small parts of large models **464**
Save and restart to isolate problem areas 464
Strategic sub-setting 465
Data reduction 466
4 Speeding up GAMS **466**
Basics **467**
Finding where excessive time is being used **467**
Screen watching and LOG file examination 468
Profile 468
Use of profile to find slow statements 469
Invoking profile 469
On the GAMS command line 470
In the IDE GAMS parameters box 470
As an internal option 470
What should the number be 470
Limiting profile output: Profiletol 470
Isolating terms in slow statements 471

© 2022 Prof. Bruce McCarl

It takes too long - searching ... 471
Why programs can be slow and their repair ... 472
Set addressing and references ... 472
Avoiding considering unnecessary cases ... 474
Calculation statements ... 474
Equation existence limited using conditionals ... 476
Equation term consideration limited using conditions ... 476
Variable specification - suppression ... 476
Watch out for incomplete suppression ... 477
Post solution report writing computations ... 477
Trading memory for time ... 477
Other speed ups ... 478
5 Memory Use Reduction in GAMS ... 478
Basics ... 479
Finding where excessive memory is being used ... 479
Screen watching and LOG file examination ... 479
Profile ... 480
Profiling to find memory hogging statements ... 481
Invoking profile ... 482
On the GAMS command line ... 482
In the IDE GAMS parameters box ... 482
As an internal option ... 482
What should the profile number be ... 482
Limiting profile output: Profiletol ... 483
Memory use dumps: Dmpsym ... 483
Looking within memory hogs to find offending term ... 485
My code won't work - searching ... 485
Causes of excessive memory use and repair ... 486
Avoiding considering unnecessary cases ... 487
Calculation statements ... 487
Equation existence using conditionals ... 487
Equation term consideration limited using conditions ... 488
Variable specification - suppression ... 488
Watch out for incomplete suppression ... 488
Memory traps to watch out for ... 489
Clearing memory of unnecessary items ... 489
Limiting memory use with HeapLimit ... 490

Chapter XIII More Language Features 490

1 Including External Files ... 490
Inclusion without arguments ... 490
$Include ... 491
Includes that cause compiler error messages ... 493
Suppressing the listing of include files ... 494
Redefining the location of include files - Idir ... 494
Include with arguments ... 494
$Batinclude ... 494
How parameter inclusion works ... 495
$Libinclude ... 497
Ldir ... 498
$Sysinclude ... 498
Sdir ... 498
Influence on LST file contents: $Oninclude and $Offinclude ... 498

© 2022 Prof. Bruce McCarl

Passing $ commands between code segments: $Onglobal and $Offglobal ... **499**
Special provision for CSV files ... **499**
$Ondelim and $Offdelim ... 499
2 Dollar Commands ... **500**
Basics ... **500**
When do dollar commands occur? ... 501
Categories of $ commands ... **502**
Commands for inclusion of comments ... 503
LST and other output file contents control ... 503
Ways of including external files ... 504
Contents dependent compilation ... 504
Alter numerical procedures used ... 506
Alter data for items ... 506
GDX file read/write ... 506
Alter compiler procedures ... 506
Cause execution of an external program ... 507
Restrict access to data ... 507
Tear apart strings ... 507
Compress and encrypt files ... 508
Handling and definition of macros ... 508
Include user defined function ... 508
Detailed description of dollar commands ... **509**
$$ ... 510
Abort ... 510
Batinclude ... 510
Call ... 510
Call.Async ... 511
Call.AsyncNC ... 512
Clear ... 512
Clearerror ... 512
Comment ... 513
Compress ... 513
Decompress ... 513
Dollar ... 514
Double ... 514
Echo, Echon ... 514
Eject ... 515
Encrypt ... 515
Eolcom ... 515
Error ... 516
Escape ... 516
Eval ... 516
Evalglobal ... 517
Evallocal ... 517
Exit ... 517
Expose ... 517
Gdxin ... 518
Gdxout ... 519
Goto ... 519
Hidden ... 520
Hide ... 520
If, If not, Ifi, Ifi not, Ife, Ife not ... 520
Ifthen, Iftheni, Ifthene, Else, Elseif, Elseifi, Elseife, Endif ... 522
Include ... 523

Inlinecom 523
Kill 523
Label 524
Libinclude 524
Lines 524
Load 524
Loaddc 526
Loaddcm 526
Loaddcr 527
Loadm 527
Loadr 527
Log 528
Macro 528
Maxcol 529
Maxgoto 529
Mincol 529
Ondelim and Offdelim 529
Ondigit and Offdigit 529
Ondollar and Offdollar 530
Ondotl and Offdotl 530
Onecho and Offecho 531
Onembedded and Offembedded 532
Onempty and Offempty 532
Onend and Offend 533
Oneolcom and Offeolcom 533
Oneps and Offeps 533
Onexpand and Offexpand 534
Onglobal and Offglobal 534
Oninclude and Offinclude 534
Oninline and Offinline 534
Onlisting and Offlisting 535
Onlocal and Offlocal 535
Onlog and Offlog 535
Onmacro and Offmacro 536
Onmargin and Offmargin 536
Onmulti and Offmulti 537
Onnestcom and Offnestcom 537
OnOrder and OffOrder 538
Onput, Onputs, Onputv, Offput 539
Onrecurse and Offrecurse 539
OnStrictSingleton and OffStrictSingleton 539
Onsymlist and Offsymlist 540
Onsymxref and Offsymxref 540
Ontext and Offtext 540
Onuellist and Offuellist 541
Onuelxref and Offuelxref 541
Onundf and Offundf 541
Onverbatim and Offverbatim 541
Onw arning and Offw arning 542
Phantom 542
Prefixpath 543
Protect 543
Purge 543
Remark 544

© 2022 Prof. Bruce McCarl

Set and Drop ... 544
Setargs ... 544
Setcomps ... 545
Setddlist ... 546
Setglobal and Dropglobal ... 546
Setenv and Dropenv ... 547
Setlocal and Droplocal ... 547
Setnames ... 548
Shift ... 548
Show ... 548
Single ... 548
SplitOption ... 549
Stars ... 549
Stop ... 549
Stitle ... 549
Sysinclude ... 550
Terminate ... 550
Title ... 550
Unload ... 550
Use205 ... 551
Use225 ... 551
Use999 ... 551
Version ... 551
Warning ... 551
3 The Option Command ... 551
Basics ... 552
Options by function ... 552
Options for control of solver choice ... 553
Options including debugging information in LST file ... 553
Options influencing LST file contents ... 553
Options influencing solver function ... 554
Other options altering GAMS settings ... 554
Options affecting data for items in memory ... 555
Options that form projections of data items ... 555
Description of options ... 555
Option itemname:d and Option itemname:d:r:c ... 555
Option itemname < or <= itemname2 ... 556
Bratio ... 557
Clear ... 557
CNS ... 557
Decimals ... 558
Dispwidth ... 558
DNLP ... 558
Domlim ... 558
DmpOpt ... 559
Dmpsym ... 559
Dualcheck ... 559
Eject ... 559
Forlim ... 559
IntVarUp ... 559
Iterlim ... 560
Kill ... 560
Limcol ... 560
Limrow ... 560

© 2022 Prof. Bruce McCarl

LP 561
MCP 561
Measure 561
MINLP 561
MIP 561
NLP 562
Oldname 562
Optca 562
Optcr 562
Profile 562
Profiletol 563
Reslim 563
RMIP 563
RMINLP 564
Savepoint 564
Seed 564
Shuffle 564
Solvelink 565
Solprint 566
Solslack 567
Solveopt 567
Solver 567
strictSingleton 568
Subsystems 568
Sys10 568
Sysout 569

Chapter XIV Advanced Language Features 569

1 Macros in GAMS 569

2 Output via Put Commands 574

Basics of put 575
Details on put related commands 577
File 578
Putdr: Pdir 578
.Pdir 578
Sending output to the LOG file 579
Sending output to the SCREEN 579
Put 579
Items within a put 580
Quoted text 580
Set elements 581
Set element names via .tl 581
Set element explanatory text via .te and .tf 582
Putting out set elements for parameters via .Tn 584
Item explanatory text via .ts 585
Numeric items 585
Parameter values 585
Model solution status attributes: .Modelstat, .Solvestat, .Tmodstat, .Tsolstat 587
Variable and equation attributes: .L and .M 588
System attributes 589
.CNS 590
.Date 590

© 2022 Prof. Bruce McCarl

.Dirsep 590
.DNLP 590
.Fe 590
.Fn 591
.Fp 591
.Gamsrelease 591
.Gstring 591
.Ifile 591
.Iline 591
.Lice1 .Lice2 591
.LP 591
.MIP 591
.MINLP 591
.NLP 591
.MCP 591
.MPEC 591
.Ofile 591
.Opage 591
.Page 591
.Pfile 591
.Platform 591
.Prline 591
.Prpage 591
.Rdate 591
.Rfile 592
.RMINLP 592
.RMIP 592
.Rtime 592
.Sfile 592
.Sstring 592
.Time 592
.Title 592
.Version 592
GAMS command line parameters 592
Write position controls 593
Skip to a specified column: @ 593
Skip to a new line: / 594
Skip to a specified row: # 595
Other positioning parameters 595
.Cc 595
.Cr 596
.Hdcc 596
.Hdcr 596
.Hdll 597
.Ll 597
.Lp 598
.Tlcc 598
.Tlll 598
.Tlcr 598
.Ws 599
Formatting of items 599
File formatting – append or overwrite 599
.Ap 599
Page formatting 600

© 2022 Prof. Bruce McCarl

.Bm - bottom margin ... 600
.Lm - left margin ... 600
.Pc - Page control ... 600
.Ps or page height ... 601
.Pw - page width ... 602
.Tm - top margin ... 602
Inserting Tabs ... 602
Adding page titles and headers ... 602
Puttl ... 603
Puthd ... 604
Putclear ... 604
Upper lower font case formatting: .Case and .Lcase ... 604
Width and decimal formatting ... 605
Global formatting ... 605
.Lw set element name width ... 606
.Nd number of decimals ... 607
.Nw number width ... 607
.Sw set indicator width ... 608
.Tw explanatory and quoted text width ... 609
Local formatting ... 610
Continuous vs fixed width ... 612
Justification ... 612
Global formatting ... 612
lj set element name justification ... 613
nj number justification ... 614
sj set indicator justification ... 615
tj explanatory and quoted text justification ... 616
Local formatting ... 618
Additional numeric display control ... 619
.nr ... 619
.nz ... 620
Putclose ... 620
Putpage ... 621
Putting out a block of text: $onput, $offput, $onputs, $onputv ... 621
Making puts conditional ... 622
Output to other programs ... 623
Put of data to a regression code ... 623
Put file for export to mapping program ... 624
Errors that arise during puts ... 625
3 Acronyms ... 625
Declaration ... 625
Usage ... 626
4 Conditional Compilation ... 627
Control variables ... 628
Establishing control variables ... 628
$Setglobal ... 628
$Setlocal ... 629
$Set ... 629
$EvalGlobal ... 630
$Evallocal ... 631
$Eval ... 632
Setting environment variables ... 633
Destroying Contol Variables ... 633

© 2022 Prof. Bruce McCarl

A problem with control variable definitions ... 633
Environment variables ... 634
Names of some system environment variables ... 634
Defining and destroying user environment variables ... 635
Augmenting environment variables ... 635
Accessing environment variable status at any point in the code: $Show ... 636
$If and $Ifi conditionals ... 637
$If and $Ifi ... 637
$Ife conditionals ... 637
Not as a modifier ... 639
$ifthen, iftheni, ifthene, else, elseif, endif conditionals ... 640
Forms of conditionals ... 643
Based on control and environment variables ... 644
Existence ... 644
Contents ... 645
Numerical Value ... 646
Based on characteristics of named item or parameter ... 648
Item type ... 648
Definition status: Declared and Defined ... 650
Set dependency: Dimension ... 650
Passed parameter existence ... 652
Based on GAMS command line parameters ... 652
Based on system characteristics ... 654
Based on error and warning checks ... 654
Based on file or directory existence ... 655
Based on put file status ... 655
Incorporating Goto: $Goto and $Label ... 656
Redefining expressions ... 657
System attributes that can be included ... 657
GAMS command line attributes that can be included ... 659
Based on user options and command line: -- // -/ /- User1-5 ... 659
Passed parameter inclusion ... 660
Control variable inclusion ... 660
Running external programs or commands ... 660
$Call ... 660
Execute ... 661
Shellexecute ... 661
$Setargs ... 661
Writing messages to LST, LOG and other files ... 661
LST File: $Abort and $Error ... 661
LOG file: $Log ... 662
Other named files: $Echo, $Offecho, $Onecho ... 662
End the job: $Exit, $Abort, $Error, $Stop, $Terminate ... 663
Longer examples ... 663
Changing model type depending on control variable ... 663
Changing form of data in model and their use ... 664
Having batincludes that deal with different data types ... 666
For more examples ... 667

Chapter XV Using GAMS Data Exchange or GDX Files 667

1 Creating a GDX file in GAMS ... 668

Command line GDX option - GDX dump of the whole problem ... 668
GDX Point Solution file ... 669

© 2022 Prof. Bruce McCarl

GDX files containing selected items ... 670
Execution time selected item GDX file creation ... 670
Compile time selected item GDX file creation ... 672
2 Inputting data from a GDX file into GAMS ... 673
Compile time imports from GDX files ... 673
Execution time GDX imports ... 675
Execute_Load ... 675
Execute_loaddc ... 676
Execute_Loadpoint ... 678
3 General notes on GDX files ... 679
4 Identifying contents of a GDX file ... 680
Identifying contents with $Load ... 680
Identifying contents with the IDE ... 681
Identifying contents with Gdxdump ... 682
Identifying differences in contents with Gdxdiff ... 685
5 Merging GDX files ... 687
6 Using GDX files to interface with other programs ... 689
Spreadsheets ... 689
GEMPACK ... 689
Other ... 689
7 Gdxcopy Making GDX files compatable ... 689
8 Writing older GDX versions with GDXCONVERT ... 691

Chapter XVI Links to Other Programs Including Spreadsheets 691

1 Executing an external program ... 692
$Call ... 693
Spaces in file names and paths ... 694
Execute ... 694
Put_utility ... 696
Timing of execution with $Call and Execute ... 699
2 Passing data from GAMS to other programs ... 701
Put file data passage ... 701
Plain text ... 702
CSV or otherwise delimited ... 703
Rutherford's CSV put: Gams2csv ... 704
GDX ... 705
Spreadsheet links ... 706
Graphics programs ... 706
Gnuplot ... 706
Gnuplot.gms ... 706
Gnuplotxyz.gms ... 706
Matlab ... 711
Spreadsheet graphics ... 711
Geographic mapping programs ... 711
GDX2ACCESS ... 711
Gdx2sqlite ... 711
Gdxrrw ... 711
Gdxviewer links: Access, Excel pivot table, Excel, CSV, GAMS include, HTML, Text files, Plots, XML ... 711
Other programs and conversions: Convert, DB2, FLM2GMS, GAMS2TBL, HTML, Latex, MPS, Oracle, XML

© 2022 Prof. Bruce McCarl

3 Passing data from other programs to GAMS 712

Including data 713

Spreadsheet links 713

Xls2gms 713

Interactive mode 713

Batch mode 715

GAMS program in Excel sheet 716

XLSDUMP 716

Database links 717

SQL: Sql2gms 717

GDX 717

Mdb2gms 717

Interactive mode 717

Batch Mode 719

Gdxrrw 720

4 API usage 720

5 Other programs: DB2, Latex, GNETGEN, Gnuplot, Matlab, MPS, NETGEN, Oracle 720

6 Customized data interchange links for spreadsheets 721

Xlexport, Xldump, Xlimport 721

Xlimport 722

Xlexport 723

Xldump 725

Gdxxrw 726

Command line parameters 727

Rng= 727

NameConv=: NC= 728

GAMS item dimension: Dim=, Rdim=, Cdim= 729

Data specification 729

Writing Text and Links 730

Set data: Set= and Dset= 731

Examples 737

Loading rows of set elements 738

Loading columns of set elements 738

Loading set elements only if they have data or text 738

Writing set elements 739

Sets and explanatory text – use of Set 739

Loading by upper left hand corner 740

Loading sets from data tables 740

Loading sets from lists with duplicates 740

Dealing with a tuple 741

Execution time set reads 741

Execution time set writes 741

Loading the set into GAMS 741

Unloading the set from GAMS 742

Parameter data: Par 743

Rearranging rows and columns 744

Variable and equation data: Equ and Var 746

Special options for reading from a spreadsheet: Skipempty= and Se= 746

Special options for writing to a spreadsheet 748

Is the workbook open or shared? 749

Merge 749

Clear 750

Filter 751

© 2022 Prof. Bruce McCarl

Special value and zero cell writing options ... 752
Epsout ... 752
Naout ... 752
Minfout ... 753
Pinfout ... 753
Undfout ... 753
Zeroout ... 753
Squeeze ... 753
Resetout ... 753
Options for reading in command line parameters ... 754
Command line parameters in a file ... 754
Parameters in a spreadsheet ... 755
Other Options ... 757
Tracing Options ... 757
Log and Logappend ... 757
Trace ... 757
Workbook performance options ... 758
Updlinks ... 758
RunMacros ... 758
Other GDXXRW Options ... 759
Debugging Gdxxrw instructions ... 759
Spreadsheet graphics ... 759
Interactively including results ... 761
Interactive calculations in a spreadsheet ... 761
Calling GAMS from GAMS ... 764
7 Using equations defined by external programs ... 766
Identifying the equations and their contents: =X= ... 766
Building the external function evaluator ... 768

Chapter XVII Controlling GAMS from External Programs 768

1 Calling GAMS from other programs ... 769
Excel spreadsheet in charge ... 769
Excel part of implementation ... 770
Defining the links through the map ... 770
Worksheets present ... 772
Inputs sheet structure ... 773
Results sheet structure ... 774
Running GAMS – the main macro ... 775
Critical user defined items ... 775
GAMS run sequence ... 775
Actions involved with executing GAMS ... 776
Examining the macros ... 776
GAMS part of implementation ... 777
Developing Excel in charge – summary steps ... 780
Excel Spawning Alternative ... 781
Compiled program in charge – Delphi ... 781
A Delphi example ... 781
Steps in application development ... 782
Passing data to GAMS ... 783
Calling GAMS ... 783
Challenges in running GAMS ... 784
Reading the GAMS solution ... 784
Web servers or programs in other languages in charge ... 785

© 2022 Prof. Bruce McCarl

2 Transferring models to other systems ... 785

Chapter XVIII Utilities included in GAMS 786

1 Posix utilities ... 786
2 Matrix Utilities ... 790
Invert ... 791
Cholesky ... 792
Eigenvalue ... 793
Eigenvector ... 794
3 GDX Utilities ... 795
CSV2GDX ... 796
Gdx2sqlite ... 797
Gdxcopy ... 798
Gdxdiff ... 798
Gdxdump ... 798
Gdxmerge ... 799
Gdxrank ... 799
Gdxrename ... 800
Gdxrrw ... 801
Gdxviewer ... 802
Gdxxrw ... 802
Gdx2access ... 802
Gdx2xls ... 802
MCFilter ... 804
MDB2GMS ... 804
SQL2GMS ... 804
Xls2gms ... 804
4 Interface utilities ... 804
Ask ... 804
Msappavail ... 807
Shellexecute ... 808
Xlstalk ... 808
5 Zip Utilities ... 809

Chapter XIX Solver Option Files 810

1 Basics ... 810
Telling a solver to look for an options file: .Optfile ... 810
Option file name ... 811
Alternative option file extention names: .Opt, .Op?, .O??, .??? ... 811
2 Option file contents ... 812
Comments: * ... 812
Option specifications ... 812
3 Option file editor ... 813
4 Writing options during a model run ... 815
5 Learning about options: Solver manuals ... 816
6 Default settings for Optfile ... 816
7 Defining a central location for the option files: Optdir ... 816
8 Transitory nature of options ... 816

© 2022 Prof. Bruce McCarl

Chapter XX Advanced Basis Usage 816

1 Basics ... 817
2 Advanced basis formation in GAMS ... 817
3 Effect of advanced basis on solution performance ... 817
4 Bratio ... 818
5 Providing a basis ... 818
Getting a basis through repeated solution ... 818
Save files ... 818
An alternative – use a GDX point file ... 819
GAMSBAS ... 820
6 Guessing at a basis ... 820
7 Problems with a basis ... 820
Symptoms and causes of a poor advanced basis ... 821
MIP ... 821
NLP ... 822

Chapter XXI Mixed Integer, Semi, and SOS Programming 822

1 Specifying types of variables ... 822
Binary variables ... 823
Integer variables ... 823
Specially ordered set variables of type 1 (SOS1) ... 824
Specially ordered set variables of type 2 (SOS2) ... 825
Semi-continuous variables ... 826
Semi-integer variables ... 827
2 Imposing priorities ... 828
3 Branch-and-Cut-and-Heuristic Facility ... 829
4 GAMS options and model attributes ... 829
Modelname.Cheat = x ... 829
Modelname.Cutoff = x ... 830
Modelname.Nodlim = x ... 830
x = Modelname.objest ... 830
Modelname.Optca=X Option Optca=X ... 831
Modelname.Optcr=X Option Optcr=X ... 831
Modelname.Optfile = 1 ... 832
Modelname.Prioropt = 1 ... 832
Modelname.Tryint = x ... 832
5 Branch and bound output ... 832
6 Nonlinear MIPs ... 833
7 Identifying the solver ... 834
MINLP ... 834
MIP ... 834
RMIP ... 834
RMINLP ... 835
8 Model termination conditions and actions ... 835
9 Things to watch out for ... 835
Default bounds ... 835

© 2022 Prof. Bruce McCarl

Ending with a gap – big default for Optcr (10%) ... 836
The nonending quest ... 836

Chapter XXII NLP and MCP Model Types 836

1 Terminology ... 837
Superbasic ... 837
Complementarity ... 837
2 Problem setup ... 837
Starting points -- initial values ... 838
Computing Derivatives ... 839
Upper and lower bounds ... 840
Scaling ... 841
Degenerate cycling blocking ... 841
Advanced bases ... 842
MCP complementarity specification ... 842
3 Output ... 843
Problem displays - limrow/limcol marking ... 843
Model setup output ... 844
Solver results ... 844
Iteration log ... 844
Termination messages ... 845
Function evaluation errors ... 845
MCP difference in Equation and Variable Solution Output ... 845
4 NLP and MCP variants ... 846
5 Solvers ... 846

Chapter XXIII Model Attributes 846

1 Attribute addressing ... 847
2 Model Attributes mainly used before solve ... 847
3 Model Attributes mainly used after solve ... 851
Marginals ... 854
Modelstat: Tmodstat ... 854
Solvestat: Tsolstat ... 855

Chapter XXIV Application Help: Model Library, Web Sites, Documentation 857

1 Libraries ... 857
GAMS model library ... 858
GAMS Test Library ... 862
GAMS Data Utilities Library ... 862
GAMS EMP Library ... 862
GAMS Financial Library ... 862
Using another model library ... 862
2 Other general documentation sources ... 864
Installation ... 864
Latest GAMS version ... 865
Solver manuals ... 865
GAMS FAQ ... 865
GAMS Wiki ... 866

© 2022 Prof. Bruce McCarl

GAMS World 866
GAMSWorld Google Group 867
Gams-List 867
Newsletter 867
Supplemental GAMS Corporation materials 868
User generated materials 868
Courses and workshops 868

Chapter XXV Compressed and encrypted files 868

Chapter XXVI Grid and Distributed Computing 870

1 Distributed Processing 870
2 Introduction to Grid Computing 873
Grid Computing language features 874
Grid_example 880

Chapter XXVII Interfacing from other languages with API 881

Chapter XXVIII Licensing 882

Index 883

© 2022 Prof. Bruce McCarl

1 McCarl GAMS User Guide

McCarl Expanded GAMS User Guide Version 24.6

by

Bruce A. McCarl
University Distinguished Professor
Regents Professor of Agricultural Economics
Texas A&M University

Alex Meeraus
Paul van der Eijk
Michael Bussieck
Steven Dirkse
Franz Nelissen
GAMS Development Corporation

© 2022 Prof. Bruce McCarl

February 28, 2016

1.1 Forword

The General Algebraic Modeling System (GAMS) is a high-level modeling system for mathematical programming problems. This document is a guide to GAMS language elements. The coverage in this document is as complete as possible for developments up to GAMS release 24.4.3 GAMS generated information on subsequent releases can be found on *Release Notes*.

The guide is designed to provide a smart document with many hyperlinks. When you click on those they will move you to related places in the document or open up example GMS files.

The document is organized into the sections identified below; an index is also present. For those wishing a printable copy as of now it is typically on C:\GAMS\win64\GAMS24.5\docs\userguides\mccarl \index.html or here.

The overall contents of the document are as follows

Quick Start Tutorial
Sets
Data Entry
Variables, Equations, Models and Solvers
Model Types and Solvers
Standard Output
Writing Models and Good Modeling Practices
Running Jobs with GAMS and the GAMS IDE
Fixing Compilation Errors
Rules for Item Names, Element Names and Explanatory Text
Including Comments
Calculating Items
Improving Output via Report Writing
Rules for Item Capitalization and Ordering
Conditionals
Control Structures
Doing a Comparative Analysis with GAMS
GAMS Command Line Parameters
Saves and Restarts
Customizing GAMS
Fixing Execution Errors
Scaling GAMS Models
Small to Large: Aid in Development and Debugging
Speeding up GAMS
Memory Use Reduction in GAMS
Including External Files
Dollar Commands
The Option Command
Output via Put Commands
Acronyms

© 2022 Prof. Bruce McCarl

Conditional Compilation
Using GAMS Data Exchange or GDX Files
Links to Other Programs Including Spreadsheets
Controlling GAMS from External Programs
Solver Option Files
Advanced Basis Usage
Mixed Integer, Semi, and SOS Programming
NLP and MCP Model Types
Model Attributes
Application Help: Model Library, Web Sites, Documentation

Many have contributed to this document beyond the authors. Erwin Kalvelagen wrote visual Basic Macros used in forming the complete document. Armin Pruessner provided comments and aid in the GDX chapter. Tony Brooke, David Kenderick, Alex Meeraus and later Ramesh Raman wrote and rewrote the earlier GAMS Users Guide which provided a foundation for this document. In addition, outside of GAMS Arne Drud of ARKI Consulting provided a number of insightful comments on the NLP chapter helping improve its contents. Gideon Kruseman of Wageningen University provided several useful comments. Rich Benjamin of FERC identified a number of errors or unclear sections in the text that we have fixed. McCarl's many students at Texas A&M and in commercial GAMS classes have also made contributions.

1.2 Introduction

The years since the 1950s have seen the rapid development of algorithms and computer codes to analyze and solve large mathematical programming problems. One important part of this growth was the development in the early 1980's of modeling systems, one of the earlier of which was the Generalized Algebraic Modeling System or GAMS - the topic of this book. GAMS is designed to

- Provide an algebraically based high-level language for the compact representation of large and complex models
- Allow changes to be made in model specifications simply and safely
- Allow unambiguous statements of algebraic relationships
- Provide an environment where model development is facilitated by subscript based expandability allowing the modeler to begin with a small data set, then after verifying correctness expand to a much broader context.
- Be inherently self documenting allowing use of longer variable, equation and index names as well as comments, data definitions etc. GAMS is designed so that model structure, assumptions, and any calculation procedures used in the report writing are documented as a byproduct of the modeling exercise in a self-contained file.
- Be an open system facilitating interface to the newest and best solvers while being solver independent allowing different solvers to be used on any given problem
- Automate the modeling process including
  - permitting data calculation;
  - verifying the correctness of the algebraic model statements;
  - checking the formulation for obvious flaws;

© 2022 Prof. Bruce McCarl

- interfacing with a solver;
- saving and submitting an advanced basis when doing related solutions;
- permitting usage of the solution for report writing.

- Permitting portability of a model formulation between computer systems allowing usage on a variety of computers ranging from PC's to workstations to super computers.
- Switching solvers is also very simple requiring changing a solver option statement or changing from using LP to using NLP.
- Facilitating import and export of data to and from other computer packages
- Allow use by groups of varying expertise
- Provide a example models that may assist modelers through provision of a model library.

This Users Guide updates and expands upon the original document by Brooke, Kenderick and Meeraus and a revision thereof by Brooke, Kenderick, Meeraus and Raman. This document unifies many system features that have occurred in the continuing system development efforts of the GAMS development Corporation with the capabilities of modern day electronic documents and computer systems.

2 Quick Start Tutorial

This book is long and detailed. Here I present a quick introductory tutorial for beginners, cross referenced to the rest of the treatment.

Basic models
Dissecting the simple models
Running the job
Examining the output
Exploiting algebra
Dissecting the algebraic model
Good modeling practices
Structure of GAMS statements, programs and the ;
Adding complexity
Why use GAMS and algebraic modeling

2.1 Basic models

In my GAMS short courses I have discovered that users approach modeling with at least three different orientations. These involve users who wish to

- Solve objective function oriented constrained optimization problems.
- Solve economically based general equilibrium problems.
- Solve engineering based nonlinear systems of equations.

In this tutorial I will use three base examples, one from each case hopefully allowing access to more than one class of user.

© 2022 Prof. Bruce McCarl

Solving an optimization problem
Solving for an economic equilibrium
Solving a nonlinear equation system

2.1.1 Solving an optimization problem

Many optimization problem forms exist. The simplest of these is the Linear Programming or LP problem. Suppose I wish to solve the optimization problem

$$
\begin{array}{llllllll}
Max & 109*X_{corn} & +90*X_{wheat} & +115*X_{Cotton} & & & \\
s.t. & X_{corn} & + X_{wheat} & + X_{Cotton} & \le 100 & (land) \\
& 6*X_{corn} & +4*X_{wheat} & +8*X_{Cotton} & \le 500 & (labor) \\
& X_{corn} & X_{wheat} & X_{Cotton} & \ge 0 & (nonnegativity)
\end{array}
$$

where this is a farm profit maximization problem with three decision variables: X_{corn} is the land area devoted to corn production, X_{wheat} is the land area devoted to wheat production, and X_{cotton} is the land area devoted to cotton production. The first equation gives an expression for total profit as a function of per acre contributions times the acreage allocated by crop and will be maximized. The second equation limits the choice of the decision variables to the land available and the third to the labor available. Finally, we only allow positive or zero acreage.

The simplest GAMS formulation of this is **optimize.gms**

```
VARIABLES             Z;
POSITIVE VARIABLES    Xcorn ,    Xwheat , Xcotton;
EQUATIONS     OBJ,  land ,  labor;
OBJ..  Z =E= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land..             Xcorn +      Xwheat +       Xcotton =L= 100;
labor..          6*Xcorn +  4 * Xwheat +  8  * Xcotton =L= 500;
MODEL farmPROBLEM /ALL/;
SOLVE farmPROBLEM USING LP MAXIMIZING Z;
```

Below after introduction of the other two examples I will dissect this formulation explaining its components.

2.1.2 Solving for an economic equilibrium

Economists often wish to solve problems that characterize economic equilibria. The simplest of these is the single good, single market problem. Suppose we wish to solve the equilibrium problem

Demand Price:	P ≥ Pd = 6 - 0.3*Qd
Supply Price:	P ≤ Ps = 1 + 0.2*Qs
Quantity Equilibrium:	Qs ≥ Qd
Non negativity:	P, Qs, Qd ≥ 0

© 2022 Prof. Bruce McCarl

where P is the market clearing price, Pd the demand curve, Qd the quantity demanded, Ps the supply curve and Qs the quantity supplied. This is a problem in 3 equations and 3 variables (the variables are P, Qd, and Qs - not Pd and Ps since they can be computed afterwards from the equality relations).

Ordinarily one would use all equality constraints for such a set up. However, I use this more general setup because it relaxes some assumptions and more accurately depicts a model ready for GAMS. In particular, I permit the case where the supply curve price intercept may be above the demand curve price intercept and thus the market may clear with a nonzero price but a zero quantity. I also allow the market price to be above the demand curve price and below the supply curve price. To insure a proper solution in such cases I also impose some additional conditions based on Walras' law.

```
Qd*( P  -  Pd )= 0     or     Qd*(Pd-(6 - 0.3*Qd))=0
Qs*( P – Ps)=0         or     Qs*(Ps-( 1 + 0.2*Qs))=0
P*(Qs-Qd)=0
```

which state the quantity demanded is nonzero only if the market clearing price equals the demand curve price, the quantity supplied is nonzero only if the market clearing price equals the supply curve price and the market clearing price is only nonzero if Qs=Qd.

The simplest GAMS formulation of this is below (**econequil.gms**). Note in this case we needed to rearrange the Ps equation so it was expressed as a greater than to accommodate the requirements of the PATH solver.

```
POSITIVE VARIABLES  P, Qd , Qs;
EQUATIONS     Pdemand,Psupply,Equilibrium;
Pdemand..     P            =g= 6 - 0.3*Qd;
Psupply..    ( 1 + 0.2*Qs) =g= P;
Equilibrium.. Qs           =g=  Qd;
MODEL PROBLEM /Pdemand.Qd,Psupply.Qs,Equilibrium.P/;
SOLVE PROBLEM USING MCP;
```

Below after introduction of the other example I will dissect this formulation explaining its components.

2.1.3 Solving a nonlinear equation system

Engineers often wish to solve a nonlinear system of equations often in a chemical equilibrium or oil refining context. Many such problem types exist. A simple form of one follows as adapted from the GAMS model library and the paper Wall, T W, Greening, D, and Woolsey , R E D, "Solving Complex Chemical Equilibria Using a Geometric-Programming Based Technique". Operations Research 34, 3 (1987). which is

```
ba * so4 = 1
baoh / ba / oh = 4.8
hso4 / so4 / h =0 .98
h * oh = 1
ba + 1e-7*baoh = so4 + 1e-5*hso4
2 * ba + 1e-7*baoh + 1e-2*h = 2 * so4 + 1e-5*hso4 + 1e-2*oh
```

© 2022 Prof. Bruce McCarl

which is a nonlinear system of equations where the variables are ba, so4, baoh, oh, hso4 and h. The simplest GAMS formulation of this is (**nonlinsys.gms**)

```
Variables ba, so4, baoh, oh, hso4, h ;
Equations r1, r2, r3, r4, b1, b2 ;
r1..  ba * so4 =e= 1 ;
r2..  baoh / ba / oh =e= 4.8 ;
r3..  hso4 / so4 / h =e= .98 ;
r4..  h * oh =e= 1 ;
b1..  ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
b2..  2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;
Model wall / all / ;
ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;
Solve wall using nlp minimizing ba;
```

2.2 Dissecting the simple models

Each of the above models is a valid running GAMS program which contains a number of common and some differentiating language elements. Let us review these elements.

Variables
Equations
.. specifications
Model
Solve
What are the .L items

2.2.1 Variables

GAMS requires an identification of the variables in a problem. This is accomplished through a VARIABLES command as reproduced below for each of the three problems.

```
VARIABLES              Z;                                    (optimize.gms)
NonNegative Variables  Xcorn ,Xwheat,Xcotton;

POSITIVE VARIABLES     P, Qd , Qs;                           (econequil.gms)

Variables              ba, so4, baoh, oh, hso4, h ;  (nonlinsys.gms)
```

The **POSITIVE** and **Nonnegative** modifiers on the variable definitions means that these variables listed thereafter are nonnegative i.e. Xcorn , Xwheat , Xcotton, P, Qd, Qs.

The use of the word VARIABLES without the POSITIVE modifier (note several other modifiers are possible as discussed in the Variables, Equations, Models and Solvers chapter) means that the named variables are unrestricted in sign as Z, ba, so4, baoh, oh, hso4, and h are above.

Notes:

© 2022 Prof. Bruce McCarl

- The general form of these statements is

  modifier variables comma or line feed specified list of variables ;

  where

  modifier is optional (positive for example)

  variable or variables is required

  a list of variables follows

  a **;** ends the statement

- This statement may be more complex including set element definitions (as we will elaborate on below) and descriptive text as illustrated in the file (**model.gms**)

```
Variables
  Tcost                    ' Total Cost Of Shipping- All Routes';
Binary Variables
  Build(Warehouse)         Warehouse Construction Variables;
Positive Variables
  Shipsw(Supplyl,Warehouse)   Shipment to warehouse
  Shipwm(Warehouse,Market) Shipment from Warehouse
  Shipsm(Supplyl,Market)   Direct ship to Demand;
Semicont Variables
  X,y,z;
```

  as discussed in the Variables, Equations, Models and Solves chapter.

- The variable names can be up to 63 characters long as discussed and illustrated in the Rules for Item Names, Element names and Explanatory Text chapter.

- GAMS is not case sensitive, thus it is **equivalent** to type the command VARIABLE as variable or the variable names XCOTTON as XcOttoN. However, there is case sensitivity with respect to the way things are printed out with the first presentation being the one used as discussed in the Rules for Item Capitalization and Ordering chapter.

- GAMS does not care about spacing or multiple lines. Also a line feed can be used instead of a comma. Thus, the following three command versions are all the same

```
POSITIVE VARIABLES  Xcorn ,Xwheat,Xcotton;
Positive Variables  Xcorn,
                    Xwheat,
                    Xcotton;
positive variables  Xcorn
                    Xwheat   ,    Xcotton;
```

2.2.1.1 What is the new Z variable in the optimization problem?

In the optimization problem I had three variables as it was originally stated but in the GAMS formulation I have four. Why? GAMS requires all optimization models to be of a special form. Namely, given the model

© 2022 Prof. Bruce McCarl

```
Maximize cx
```

It must be rewritten as

```
Maximize        R
                R=CX
```

where R is a variable unrestricted in sign. This variable can be named however you want it named (in the above example case Z). There always must be at least one of these in every problem which is the objective function variable and it must be named as the item to maximize or minimize.

Thus in a problem one needs to declare a new unrestricted variable and define it through an equation. In our optimization example (**optimize.gms**) we declared Z as a Variable (not a Positive Variable), then we declared and specified an equation setting Z equal to the objective function expression and told the solver to maximize Z,

```
VARIABLES             Z;
EQUATIONS     OBJ,  land ,  labor;
OBJ.. Z =E=
      109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
SOLVE PROBLEM USING LP MAXIMIZING Z;
```

Note users do not always have to add such an equation if there is a variable in the model that is unrestricted in sign that can be used as the objective function. For example the equation solving case (**nonlinsys.gms**) uses a maximization of ba as a dummy objective function (as further discussed below the problem is really designed to just solve the nonlinear system of equations and the objective is just there because the model type used needed one).

2.2.2 Equations

GAMS requires that the modeler name each equation, which is active in the optimization model. Later each equation is specified using the .. notation as explained just below. These equations must be named in an EQUATION or EQUATIONS instruction. This is used in each of the example models as reproduced below

```
EQUATIONS     OBJ,  land ,  labor;                  (optimize.gms)

EQUATIONS     PDemand,PSupply, Equilibrium;         (econequil.gms)

Equations r1, r2, r3, r4, b1, b2 ;                  (nonlinsys.gms)
```

Notes:

- The general form of these statements are

  Equations comma or line feed specified list of equations ;

  where

  equation or equations is required

© 2022 Prof. Bruce McCarl

a list of equations follows

a **;** ends the statement

- In optimization models the objective function is always defined in one of the named equations.
- This statement may be more complex including set element definitions (as we will elaborate on below) and descriptive text as illustrated in the file (**model.gms**)

```
EQUATIONS
         TCOSTEQ             TOTAL COST ACCOUNTING EQUATION
         SUPPLYEQ(SUPPLYL)   LIMIT ON SUPPLY AVAILABLE AT A SUPPLY POINT
         DEMANDEQ(MARKET)    MINIMUM REQUIREMENT AT A DEMAND MARKET
         BALANCE(WAREHOUSE)  WAREHOUSE SUPPLY DEMAND BALANCE
         CAPACITY(WAREHOUSE) WAREHOUSE CAPACITY
         CONFIGURE           ONLY ONE WAREHOUSE;
```

  as discussed in the Variables, Equations, Models and Solves chapter.

- The equation names can be up to 63 characters long as discussed and illustrated in the Rules for Item Names, Element Names and Explanatory Text chapter.

2.2.3 .. specifications

The GAMS equation specifications actually consist of two parts. The first part naming equations, was discussed just above. The second part involves specifying the exact algebraic structure of equations.

This is done using the **..** notation. In this notation we give the equation name followed by a **..** then the exact equation type as it should appear in the model. The equation type specification involves use of a special syntax to tell the exact form of the relation involved. The most common of these are (see the Variables, Equations, Models and Solves chapter for a complete list):

```
=E= is used to indicate an equality relation
=L= indicates a less than or equal to relation
=G= indicates a greater than or equal to relation
```

This is used in each of the example models where a few of the component equations are reproduced below

```
OBJ..  Z =E= 109*Xcorn + 90*Xwheat + 115*Xcotton;      (optimize.gms)
land..          Xcorn +    Xwheat +     Xcotton =L= 100;

Pdemand..     P           =g= 6 - 0.3*Qd;              (econequil.gms)

r1..  ba * so4 =e= 1 ;                                (nonlinsys.gms)
```

Notes:

- The general form of these statements is

© 2022 Prof. Bruce McCarl

Equationname .. algebra1 equationtype algebra2 ;

where

an equation with that name must have been declared (have appeared in and equation statement)

the .. appears just after the equation name

the algebraic expressions algebra1 and algebra2 can each be a mixture of variables, data items and constants

the equationtype is the =E=, =L=, and =G= discussed above.

a ; ends the statement

- All equations must be specified in .. notation before they can be used.
- Some model equations may be specified in an alternative way by including upper or lower bounds as discussed in the Variables, Equations, Models and Solves chapter.
- .. specification statements may be more complex including more involved algebra as discussed later in this tutorial and in the Calculating Items chapter.
- It may be desirable to express equations as only being present under some conditions as discussed later in this tutorial and in the Conditionals chapter.

2.2.4 Model

Once all the model structural elements have been defined then one employs a MODEL statement to identify models that will be solved. Such statements occur in the each of the three example models:

```
MODEL farmPROBLEM /ALL/;                                    (optimize.gms)

MODEL PROBLEM /Pdemand.Qd, Psupply.Qs,Equilibrium.P/;       (econequil.gms)

Model wall / all / ;                                        (nonlinsys.gms)
```

Notes:

- The general form of these statements are

  Model modelname optional explanatory text / model contents/ ;

  where

  Model or models is required
  a modelname follows that can be up to 63 characters long as discussed in the Rules for Item Names, Element names and Explanatory Text chapter
  the optional explanatory text is up to 255 characters long as discussed in the Rules for Item Names, Element names and Explanatory Text chapter
  the model contents are set off by beginning and ending slashes and can either be the keyword all including all equations, a list of equations, or a list of equations and

© 2022 Prof. Bruce McCarl

complementary variables. Each of these is discussed in the following bullets.

a **;** ends the statement

- In the Model Statement in the model contents field

  Using /ALL/ includes all the equations.

  One can list equations in the model statement like that below.

  ```
  MODEL FARM /obj, Land,labor/;
  ```

  and one does not need to list all the equations listed in the Equations statements. Thus, in (**optimize.gms**), one could omit the constraints called labor from the model

  ```
  MODEL ALTPROBLEM / obj,land/;
  ```

- The equilibrium problems are solved as Mixed complementarity problems (MCP) and require a special variant of the Model statement. Namely in such problems there are exactly as many variables as there are equations and each variable must be specified as being complementary with one and only one equation. The model statement expresses these constraints indicating the equations to be included followed by a period(.) and the name of the associated **complementary variables** as follows

  ```
  MODEL PROBLEM /Pdemand.Qd, Psupply.Qs,Equilibrium.P/; (econequil.gms)
  ```

  which imposes the complementary relations from our equilibrium problem above.

- All equations in the model which are named and any data included must have been specified in .. notation before this model can be used (in a later solve statement).

- Users may create several models in one run each containing a different set of equations and then solve those models and separately.

2.2.5 Solve

Once one believes that the model is ready in such that it makes sense to find a solution for the variables then the solve statement comes into play. The SOLVE statement causes GAMS to use a solver to optimize the model or solve the embodied system of equations.

```
SOLVE farmPROBLEM USING LP MAXIMIZING Z;          (optimize.gms)

SOLVE PROBLEM USING MCP;                          (econequil.gms)

Solve wall using nlp minimizing ba;               (nonlinsys.gms)
```

Notes:

- The general forms of these statements for models with objective functions are

  Solve modelname using modeltype maximizing variablename ;

  Solve modelname using modeltype minimizing variablename ;

© 2022 Prof. Bruce McCarl

and for models without objective functions is

Solve modelname using modeltype;

where

Solve is required
a modelname follows that must have already been given this name in a Model statement
using is required
the modeltype is one of the known GAMS model types where

models with objective functions are

LP for linear programming

NLP for nonlinear programming

MIP for mixed integer programming

MINLP for mixed integer non linear programming

plus RMIP, RMINLP, DNLP, MPEC as discussed in the chapter on Model Types and Solvers.

models without objective functions are

MCP for mixed complementary programming

CNS for constrained nonlinear systems

maximizing or minimizing is required for all optimization problems (not MCP or CNS problems)

a variablename to maximize or minimize is required for all optimization problems (not MCP or CNS problems) and must match with the name of a variable defined as free or just as a variable.

a ; ends the statement

- The examples statement solve three different model types

  a linear programming problem ("using LP").

  a mixed complementary programming problem ("using MCP").

  a non linear programming problem ("using NLP").

- GAMS does not directly solve problems. Rather it interfaces with external solvers developed by other companies. This requires special licensing arrangements to have access to the solvers. It also requires that, for the user to use a particular solver, it already must have been interfaced with GAMS. A list of the solvers currently interfaced is covered in the Model Types and Solvers chapter.

2.2.5.1 Why does my nonlinear equation system maximize something?

The nonlinear equation system chemical engineering problem in the GAMS formulation was expressed as a nonlinear programming (NLP) optimization model in turn requiring an objective function. Actually this is somewhat older practice in GAMS as the constrained nonlinear system (CNS) model type was added after this example was initially formulated. Thus, one could modify the model type to solve constrained nonlinear system yielding the same solution using

© 2022 Prof. Bruce McCarl

```
Solve wall using mcp;                    (nonlinsyscns.gms)
```

However, the CNS model type can only be solved by select solvers and cannot incorporate integer variables. Formulation as an optimization problem relaxes these restrictions allowing use of for example the MINLP model type plus the other NLP solvers. Such a formulation involves the choice of a convenient variable to optimize which may not really have any effect since a feasible solution requires all of the simultaneous equations to be solved. Thus, while ba is maximized, there is no inherent interest in attaining its maximum; it is just convenient.

2.2.6 What are the .L items

In the nonlinear equation system chemical engineering GAMS formulation a line was introduced which is

```
ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;    (nonlinsys.gms)
```

This line provides a starting point for the variables in the model. In particular the notation variablename.l=value is the way one introduces a starting value for a variable in GAMS as discussed in the chapter on NLP and MCP Model Types. Such a practice can be quite important in achieving success and avoiding numerical problems in model solution (as discussed in the Fixing Execution Errors chapter).

Notes:

- One may also need to introduce lower (variablename.lo=value) and upper (variablename.up=value) bounds on the variables as also discussed in the Fixing Execution Errors chapter.
- The .l, .lo and .up appendages on the variable names are illustrations of variable attributes as discussed in the Variables, Equations, Models and Solves chapter.
- The = statements setting the variable attributes to numbers are the first example we have encountered of a GAMS assignment statement as extensively discussed in the Calculating Items chapter.

2.3 Running the job

GAMS is a two pass program. One first uses an editor to create a file nominally with the extension GMS which contains GAMS instructions. Later when the file is judged complete one submits that file to GAMS. In turn, GAMS executes those instructions causing calculations to be done, solvers to be used and a solution file of the execution results to be created. Two alternatives for submitting the job exist the traditional command line approach and the IDE approach.

Command line approach
IDE approach

2.3.1 Command line approach

The basic procedure involved for running command line GAMS is to create a file (nominally myfilename.gms where myfilename is whatever is a legal name on the operating system being used)

© 2022 Prof. Bruce McCarl

with a text editor and when done run it with a DOS or UNIX or other operating system command line instruction like

```
GAMS trnsport
```

where trnsport.gms is the file to be run. Note the gms extension may be omitted and GAMS will still find the file.

The basic command line GAMS call also allows a number of arguments as illustrated below

```
GAMS TRNSPORT pw=80 ps=9999 s=mysave
```

which sets the page width to 80, the page length to 9999 and saves work files. The full array of possible command line arguments is discussed in the Command Line Parameters chapter. When GAMS is run the answers are placed in the LST file. Namely if the input file of GAMS instructions is called myfile.gms then the output will be on myfile.LST.

2.3.2 IDE approach

Today with the average user becoming oriented to graphical interfaces it was a natural development to create the GAMSIDE or IDE for short. The IDE is a GAMS Corporation product providing an Integrated Development Environment that is designed to provide a Windows graphical interface to allow for editing, development, debugging, and running of GAMS jobs all in one program. I will not cover IDE usage in this tutorial and rather refer the reader to the tutorial on IDE usage that appears in the chapter on Running Jobs with GAMS and the GAMS IDE. When the IDE is run there is again the creation of the LST file. Namely if the input file of GAMS instructions is called myfile.gms then the output will be on myfile.LST.

2.4 Examining the output

When a GAMS file is run then GAMS in turn creates a LST file of problem results. One can edit the LST file in either the IDE or with a text editor to find any error messages, solution output, report writing displays etc. In turn one can also reedit the GMS file if there were need to fix anything or alter the model contents and rerun with GAMS until a satisfactory result is attained. Now let us review the potential elements of the LST file.

Echo print
Symbol list and cross reference maps
Execution output
Generation listing
Solver report

2.4.1 Echo print

The first item contained within the LST file is the echo print. The echo print is simply a numbered copy of the instructions GAMS received in the GMS input file. For example, in the LST file segment immediately below is the portion associated with the GAMS instructions in **optimize.gms**.

```
     3  VARIABLES               Z;v
```

© 2022 Prof. Bruce McCarl

```
 4  POSITIVE VARIABLES    Xcorn ,    Xwheat , Xcotton;
 5  EQUATIONS     OBJ,  land ,  labor;
 6  OBJ..  Z =E= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
 7  land..             Xcorn +      Xwheat +       Xcotton =L= 100;
 8  labor..          6*Xcorn +  4 * Xwheat +  8  * Xcotton =L= 500;
 9  MODEL farmPROBLEM /ALL/;
10  SOLVE farmPROBLEM USING LP MAXIMIZING Z;
```

Notes:

- The echo print is of the same character for all three examples so I only include the **optimize.gms** LST file echo print here.
- The echo print can incorporate lines from other files if include files are present as covered in the Including External Files chapter.
- The echo print can be partially or fully suppressed as discussed in the Standard Output chapter.
- The numbered echo print often serves as an important reference guide because GAMS reports the line numbers in the LST file where solves or displays were located as well as a the position of any errors that have been encountered.

2.4.1.1 Incidence of compilation errors

GAMS requires strict adherence to language syntax. It is very rare for even experienced users to get their syntax exactly right the first time. GAMS marks places where syntax does not correspond exactly as compilation errors in the echo print listing. For example I present the echo print from a syntactically incorrect variant of the economic equilibrium problem. In that example (**econequilerr.gms**) I have introduced errors in the form of a different spelling of the variable named Qd between line's 1, 3, 5 and 6 spelling it as Qd in line 1 and Qdemand in the other three lines. I also omit a required ; in line 4.

```
   1  POSITIVE VARIABLES  P, Qd , Qs;
   2  EQUATIONS     PDemand,PSupply, Equilibrium;
   3  Pdemand..     P            =g= 6 - 0.3*Qdemand;
****                                        $140
   4  Psupply..    ( 1 + 0.2*Qs) =g= P
   5  Equilibrium.. Qs           =g=  Qdemand;
****           $409
   6  MODEL PROBLEM /Pdemand.Qdemand, Psupply.Qs,Equilibrium.P/;
****                             $322
   7  SOLVE PROBLEM USING MCP;
****                        $257

Error Messages
140  Unknown symbol
257  Solve statement not checked because of previous errors
322  Wrong complementarity pair. Has to be equ.var.
409  Unrecognizable item - skip to find a new statement
       looking for a ';' or a key word to get started again
```

The above echo print contains the markings relative to the compiler errors. A compiler error message consists of three important elements. First a marker **** appears in line just beneath the line where an

© 2022 Prof. Bruce McCarl

error occurred. Second a $ is placed in the LST file just underneath the position in the above line where the error occurred. Third a numerical code is entered just after the $ which cross-references to a list appearing later in the LST file of the heirs encountered and a brief explanation of their cause sometimes containing a hint on how to repair the error.

Notes:

- The above messages and markings show GAMS provides help in locating errors and givies clues as to what's wrong. Above there are error markings in every position where Qdemand appears indicating that GAMS does not recognize the item mainly because it does not match with anything within the variable or other declarations above. It also marks the 409 error in the Equilibrium equation just after the missing ; and prints a message that indicates that a ; may be the problem.
- The **** marks all error messages whether they be compilation or execution errors. Thus, one can always search in the LST file for the **** marking to find errors.
- It is recommended that users do not use lines with **** character strings in the middle of their code (say in a comment as can be entered by placing an * in column 1—see the Including Comments chapter) but rather employ some other symbol.
- The example illustrates error proliferation. In particular the markings for the errors 140, 322 and 409 identify the places mistakes were made but the error to 257 does not mark a mistake. Also while the 140 and 322 mark mistakes, the real mistake may be that in line 1 where Qd should have been spelled as Qdemand. It is frequent in GAMS that a declaration error causes a lot of subsequent errors.
- In this case only two corrections need to be made to repair the file. One should spell Qd in line 1 as Qdemand or conversely change all the later references to Qd. One also needs to add a semi colon to the end of line 4.
- The IDE contains a powerful navigation aid which helps users directly jump from error messages into the place in the GMS code where the error message occurs as discussed in the Running Jobs with GAMS and the GAMS IDE chapter.
- When multiple errors occur in a single position, GAMS cannot always locate the $ just in the right spot as that spot may be occupied.
- New users may find it desirable to reposition the error message locations so the messages appear just below the error markings as discussed in the Fixing Compilation Errors chapter.
- Here I have only presented a brief introduction to compilation error discovery. The chapter on Fixing Compilation Errors goes substantially further and covers through example a number of common error messages received and their causes.

2.4.2 Symbol list and cross reference maps

The next component of the LST file is the symbol list and cross-reference map. These may or not be present as determined by the default settings of GAMS on your system. In particular, while these items appear by default when running command line GAMS they are suppressed by default when running the IDE.

The more useful of these outputs is the symbol list that contains an alphabetical order all the variables, equations, models and some other categories of GAMS language classifications that I have not yet discussed along with their optional explanatory text. These output items will not be further covered in this tutorial, but are covered in the Standard Output chapter.

© 2022 Prof. Bruce McCarl

2.4.3 Generation listing

Once GAMS has successfully compiled and executed then any solve statements that are present will be implemented. In particular, the GAMS main program generates a computer readable version of the equations in the problem that it in turn passes on to whatever third party solver is going to be used on the model. During this so called model generation phase GAMS creates output

- Listing the specific form of a set of equations and variables,
- Providing a summary of the total model structure, and
- If encountered, detailing any numerical execution errors that occurred in model generation.

Each of these excepting execution errors will be discussed immediately below. Model generation time execution errors are discussed in the Fixing Execution Errors chapter.

2.4.3.1 Equation listing

When GAMS generates the model by default the first three equations for each named equation will be generated. A portion of the output (just that for the first two named equations) for the each for the three example models is

```
Equation Listing    SOLVE farmPROBLEM Using LP From line 10
---- OBJ  =E=
OBJ..  Z - 109*Xcorn - 90*Xwheat - 115*Xcotton =E= 0 ; (LHS = 0)
---- land  =L=
land..  Xcorn + Xwheat + Xcotton =L= 100 ; (LHS = 0)

Equation Listing    SOLVE wall Using NLP From line 28
---- PDemand  =G=
PDemand..  P + 0.3*Qd =G= 6 ; (LHS = 0, INFES = 6 ***)
---- PSupply  =G=
PSupply..  - P + 0.2*Qs =G= -1 ; (LHS = 0)

Equation Listing    SOLVE PROBLEM Using MCP From line 7
---- r1  =E=
r1..  (1)*ba + (1)*so4 =E= 1 ; (LHS = 1)
---- r2  =E=
r2..  - (1)*ba + (1)*baoh - (1)*oh =E= 4.8 ; (LHS = 1, INFES = 3.8
***)
```

Notes:

- The first part of this output gives the words **Equation Listing** followed by the word Solve, the name of the model being solved and the line number in the echo print file where the solve associated with this model generation appears.
- The second part of this output consists of the marker ---- followed by the name of the equation with the relationship type (=L=, =G=, =E= etc).
- When one wishes to find this LST file component, one can search for the marker ---- or the string Equation Listing. Users will quickly find ---- marks other types of output like that from display statements.

© 2022 Prof. Bruce McCarl

- The third part of this output contains the equation name followed by a .. and then a listing of the equation algebraic structure. In preparing this output, GAMS collects all terms involving variables on the left hand side and all constants on the right hand side. This output component portrays the equation in linear format giving the names of the variables that are associated with nonzero equation terms and their associated coefficients.
- The algebraic structure portrayal is trailed by a term which is labeled LHS and gives at evaluation of the terms involving endogenous variables evaluated at their starting points (typically zero unless the .L levels were preset). A marker INFEAS will also appear if the initial values do not constitute a feasible solution.
- The equation output is a correct representation of the algebraic structure of any linear terms in the equation and a local representation containing the first derivatives of any nonlinear terms. The nonlinear terms are automatically encased in parentheses to indicate a local approximation is present. For example in the non-linear equation solving example the first equation is algebraically structured as

```
ba * so4 = 1
```

but the equation listing portrays this as additive

```
---- r1  =E=
r1..  (1)*ba + (1)*so4 =E= 1 ; (LHS = 1)
```

which the reader can verify as the first derivative use of the terms evaluated around the starting point (ba=1,so4=1).

More details on how the equation list is formed and controlled in terms of content and length are discussed in the Standard Output chapter while more on nonlinear terms appears in the NLP and MCP Model Types chapter.

2.4.3.2 Variable listing

When GAMS generates the model by default the data for the first three cases in existence under each named variable will be generated. A portion of the output (just that for the first two named variables) for the each for the three example models is

```
Column Listing      SOLVE farmPROBLEM Using LP From line 10
---- Z
Z
                (.LO, .L, .UP = -INF, 0, +INF)
        1       OBJ
---- Xcorn
Xcorn
                (.LO, .L, .UP = 0, 0, +INF)
     -109       OBJ
        1       land
        6       labor

Column Listing      SOLVE PROBLEM Using MCP From line 7
---- P
P
                (.LO, .L, .UP = 0, 0, +INF)
        1       PDemand
```

© 2022 Prof. Bruce McCarl

```
       -1       PSupply
---- Qd
Qd
                (.LO, .L, .UP = 0, 0, +INF)
        0.3     PDemand
       -1       Equilibrium

Column Listing    SOLVE wall Using NLP From line 28
---- ba
ba
                (.LO, .L, .UP = -INF, 1, +INF)
       (1)      r1
      (-1)      r2
        1       b1
        2       b2
---- so4
so4
                (.LO, .L, .UP = -INF, 1, +INF)
       (1)      r1
      (-1)      r3
       -1       b1
       -2       b2
```

Notes:

- The first part of this output gives the words **Column Listing** followed by the word Solve, the name of the model being solved and the line number in the echo print file where the solve associated with this model generation appears.
- The second part of this output consists of the marker ---- followed by the name of the variable.
- When one wishes to find this LST file component, one can search for the marker ---- or the string Column Listing. Users will quickly find ---- marks other types of output like that from display statements.
- The third part of this output contains the variable name followed by (.LO, .L, .UP = lower bound, starting level, upper bound) where

  lower bound gives the lower bound assigned to this variable (often zero)

  starting level gives the starting point assigned to this variable (often zero)

  upper bound gives the lower bound assigned to this variable (often positive infinity + INF).
- The fourth part of this output gives the equation names in which this variable appears with a nonzero term and the associated coefficients.
- The output is a correct representation of the algebraic structure of any linear terms in the equations where the variable appears and a local representation containing the first derivatives of any nonlinear terms. The nonlinear terms are automatically encased in parentheses to indicate a local approximation is present just analogous to the portrayals in the equation listing section just above.

  More details on how the variable list is formed and controlled in terms of content and length are discussed in the Standard Output chapter while more on nonlinear terms appears in the NLP and MCP Model Types chapter.

© 2022 Prof. Bruce McCarl

2.4.3.3 Model statistics

GAMS also creates an output summarizing the size of the model as appears just below from the non-linear equation solving example nonlinsys.gms. This gives how many variables of equations and nonlinear terms are in the model along with some additional information. For discussion of the other parts of this output see the Standard Output and NLP and MCP model types chapters.

```
MODEL STATISTICS

BLOCKS OF EQUATIONS     6        SINGLE EQUATIONS       6
BLOCKS OF VARIABLES     6        SINGLE VARIABLES       6
NON ZERO ELEMENTS      20        NON LINEAR N-Z        10
DERIVATIVE POOL         6        CONSTANT POOL          8
CODE LENGTH            89
```

2.4.4 Execution output

The next, usually minor, element of the GAMS LST file is execution report. Typically this will involve

- A report of the time it takes GAMS to execute any statements between the beginning of the program and the first solve (or in general between solves),
- Any user generated displays of data; and
- If present, a list of numerical execution errors that arose.

I will not discuss the nature of this output here, as it is typically not a large concern of new users. Display statements will be discussed later within this tutorial and are discussed in the Improving Output via Report Writing chapter. Execution errors and their markings are discussed in the Fixing Execution Errors chapter.

2.4.5 Solver report

The final major component of the LST file is the solution output and consists of a summary and then a report of the solutions for variables and equations. Execution error reports may also appear in nonlinear models as discussed in the Fixing Execution Errors Chapter.

2.4.5.1 Solution summary

The solution summary contains

- the marker S O L V E S U M M A R Y;
- the model name, objective variable name (if present), optimization type (if present), and location of the solve (in the echo print);
- the solver name;
- the solve status in terms of solver termination condition;
- the objective value (if present);
- some cpu time expended reports;
- a count of solver execution errors; and

© 2022 Prof. Bruce McCarl

- some solver specific output.

The report from the non-linear equation solving example **nonlinsys.gms** appears just below.

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

     MODEL   wall                OBJECTIVE  ba
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  CONOPT              FROM LINE  28

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE                1.0000

 RESOURCE USAGE, LIMIT          0.090      1000.000
 ITERATION COUNT, LIMIT         5         10000
 EVALUATION ERRORS              0             0

    C O N O P T 2   Windows NT/95/98  version 2.071J-011-046
    Copyright (C)   ARKI Consulting and Development A/S
                    Bagsvaerdvej 246 A
                    DK-2880 Bagsvaerd, Denmark
 Using default control program.
 ** Optimal solution. There are no superbasic variables.
```

More on this appears in the Standard Output chapter.

2.4.5.2 Equation solution report

The next section of the LST file is an equation by equation listing of the solution returned to GAMS by the solver. Each individual equation case is listed. For our three examples the reports are as follows

```
                     LOWER     LEVEL     UPPER    MARGINAL
---- EQU OBJ           .         .         .         1.000
---- EQU land        -INF     100.000   100.000    52.000
---- EQU labor       -INF     500.000   500.000     9.500

                     LOWER     LEVEL     UPPER    MARGINAL
---- EQU PDemand      6.000     6.000     +INF      10.000
---- EQU PSupply     -1.000    -1.000     +INF      10.000
---- EQU Equilibri~    .         .        +INF       3.000

                     LOWER     LEVEL     UPPER    MARGINAL
---- EQU r1           1.000     1.000     1.000      0.500
---- EQU r2           4.800     4.800     4.800       EPS
---- EQU r3           0.980     0.980     0.980  4.9951E-6
---- EQU r4           1.000     1.000     1.000  2.3288E-6
---- EQU b1             .         .         .        0.499
---- EQU b2             .         .         .    2.5676E-4
```

© 2022 Prof. Bruce McCarl

The columns associated with each entry have the following meaning,

- Equation marker ----
- EQU - Equation identifier
- Lower bound (.lo) – RHS on =G= or =E= equations
- Level value (.l) – value of Left hand side variables. Note this is not a slack variable but inclusion of such information is discussed in the Standard Output chapter.
- Upper bound (.up) – RHS on =L= or =E= equations
- Marginal (.m) – dual variable, shadow price or in MCPs only complementary variable value (See the NLP and MCP chapter)

Notes:

- The numbers are printed with fixed precision, but the values are returned within GAMS have full machine accuracy.
- The single dots '.' represent zeros.
- If present 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 marginals are zero for basic variables, and nonzero for others.
- 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.
- For models that are not solved to optimality, some items may additionally be marked with the following flags.

Flag	Description
Infes	The item is infeasible. This mark is made for any entry whose level value is not between the upper and lower bounds.
Nopt	The item 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.

- The marginal output generally does not have much meaning in an MCP or CNS model.

2.4.5.3 Variable solution report

The next section of the LST file is a variable by variable listing of the solution returned to GAMS by the solver. Each individual variable case is listed. For our three examples the reports are as follows

```
                           LOWER      LEVEL      UPPER     MARGINAL

---- VAR Z                 -INF    9950.000      +INF         .
---- VAR Xcorn              .        50.000      +INF         .
---- VAR Xwheat             .        50.000      +INF         .
```

© 2022 Prof. Bruce McCarl

```
---- VAR Xcotton          .          .        +INF     -13.000

                          LOWER      LEVEL      UPPER    MARGINAL
---- VAR P                  .        3.000      +INF        .
---- VAR Qd                 .       10.000      +INF        .
---- VAR Qs                 .       10.000      +INF        .

                          LOWER      LEVEL      UPPER    MARGINAL
---- VAR ba                -INF       1.000      +INF        .
---- VAR so4               -INF       1.000      +INF        .
---- VAR baoh              -INF       4.802      +INF        .
---- VAR oh                -INF       1.000      +INF        .
---- VAR hso4              -INF       0.980      +INF        .
---- VAR h                 -INF       1.000      +INF        .
```

The columns associated with each entry have the following meaning,

- Variable marker ----
- VAR - Variable identifier
- Lower bound (.lo) – often zero or minus infinity
- Level value (.l) – solution value.
- Upper bound (.up) – often plus infinity
- Margninal (.m) – reduced cost or in MCPs only slack in complementary equations (See the NLP and MCP chapter)

Notes:

- The numbers are printed with fixed precision, but the values are returned within GAMS have full machine accuracy.
- The single dots '.' represent zeros.
- If present 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 marginals are zero for basic variables, and nonzero for others.
- 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.
- For models that are not solved to optimality, some items may additionally be marked with the following flags.

Flag	Description
Infes	The item is infeasible. This mark is made for any entry whose level value is not between the upper and lower bounds.
Nopt	The item 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.

© 2022 Prof. Bruce McCarl

2.5 Exploiting algebra

By its very nature GAMS is an algebraic language. The above examples and discussion are not totally exploitive of the algebraic capabilities of GAMS. Now let me introduce more of the GAMS algebraic features.

Equation writing – sums
Revised algebra exploiting optimization example
Revised equilibrium example

2.5.1 Equation writing – sums

GAMS is fundamentally built to allow exploitation of algebraic features like summation notation. Specifically suppose xi is defined with three elements

$$\sum_i X_i = X_1 + X_2 + X_3$$

Algebra

This can be expressed in GAMS as

```
z = SUM(I, X(I));
```

where

I	is a set in GAMS
z	is a scalar or variable
X(I)	is a parameter or variable defined over set I

and the sum automatically treats all cases of I.

Such an expression can be included either in a either a model equation .. specification or in an item to be calculated in the code. Let me now remake the first 2 examples, better exploiting the GAMS algebraic features.

2.5.2 Revised algebra exploiting optimization example

The optimization example is as follows

© 2022 Prof. Bruce McCarl

$$
\begin{array}{lrrrrl}
Max & 109*X_{corn} & +90*X_{wheat} & +115*X_{Cotton} & & \\
s.t. & X_{corn} & +X_{wheat} & +X_{Cotton} & \le 100 & (land) \\
& 6*X_{corn} & +4*X_{wheat} & +8*X_{Cotton} & \le 500 & (labor) \\
& X_{corn} & X_{wheat} & X_{Cotton} & \ge 0 & (nonnegativity)
\end{array}
$$

This is a special case of the general resource allocation problem that can be written as

$$
\begin{array}{lrcll}
Max & \sum_j C_j X_j & & & \\
s.t. & \sum_j a_{ij} X_j & \le & b_i & for\ all\ i \\
& X_j & \ge & 0 & for\ all\ j
\end{array}
$$

where

j =	{	corn		wheat	cotton	}
i =	{	land		labor	}	
x_j =	{	X_{corn}	X_{wheat}	X_{cotton}	}	
c_j =	{	109	90	115	}	
a_{ij} =		1		1		1
		6		4		8
b_i =	{	100		500	}'	

Such a model can be cast in GAMS as (**optalgebra.gms**)

```
SET         j      /Corn,Wheat,Cotton/
            i      /Land ,Labor/;
PARAMETER
  c(j)          / corn     109    ,wheat   90 ,cotton    115/
  b(i)          /land 100 ,labor 500/;
TABLE a(i,j)
             corn    wheat    cotton
land           1        1        1
labor          6        4        8      ;
POSITIVE VARIABLES    x(j);
VARIABLES                PROFIT            ;
```

© 2022 Prof. Bruce McCarl

```
EQUATIONS                 OBJective         ,
                          constraint(i) ;
OBJective..       PROFIT=E=   SUM(J,(c(J))*x(J)) ;
constraint(i)..            SUM(J,a(i,J) *x(J))  =L= b(i);
MODEL    RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

I will dissect the GAMS components after presenting the other example.

2.5.3 Revised equilibrium example

The economic equilibrium model was of the form

Demand Price:	P > Pd = 6 - 0.3*Qd
Supply Price:	P < Ps = 1 + 0.2*Qs
Quantity Equilibrium:	Qs > Qd
Non negativity	P, Qs, Qd > 0

and is a single commodity model. Introduction of multiple commodities means that we need a subscript for commodities and consideration of cross commodity terms in the functions. Such a formulation where c depicts commodity can be presented as

Demand Price for c:	$P_c \geq Pd_c = Id_c - \sum_{cc} Sd_{c,cc} * Qd_{cc}$	for all c
Supply Price for c:	$P_c \leq Ps_c = Is_c + \sum_{cc} Ss_{c,cc} * Qs_{cc}$	for all c
Quantity Equil. for c:	$Qs_c \geq Qd_c$	for all c
Non negativity	$P_c , Qd_c , Qs_c \geq 0$	for all c

where

- P_c is the price of commodity c
- Qd_c is the quantity demanded of commodity c
- Pd_c is the price from the inverse demand curve for commodity c
- Qs_c is the quantity supplied of commodity c
- Ps_c is the price from the inverse supply curve for commodity c
- cc is an alternative index to the commodities and is equivalent to c
- Id_c is the inverse demand curve intercept for c
- $Sd_{c,cc}$ is the inverse demand curve slope for the effect of buying one unit of commodity cc on the demand price of commodity c. When c=cc this is an own commodity effect and when c≠cc then this is a cross commodity effect.
- Is_c is the inverse supply curve intercept for c
- $Ss_{c,cc}$ is the inverse supply curve slope for the effect of supplying one unit of commodity cc on the supply price of commodity c. When c=cc this is an own commodity effect and

© 2022 Prof. Bruce McCarl

when c≠cc then this is a cross commodity effect.

An algebraic based GAMS formulation of this is (**econequilalg.gms**)

```
Set commodities /corn,wheat/;
Set curvetype /Supply,demand/;
Table intercepts(curvetype,commodities)
                  corn   wheat
         demand    4       8
         supply    1       2;
table slopes(curvetype,commodities,commodities)
                   corn  wheat
     demand.corn   -.3    -.1
     demand.wheat  -.07   -.4
     supply.corn    .5     .1
     supply.wheat   .1     .3     ;
POSITIVE VARIABLES  P(commodities)
                    Qd(commodities)
                    Qs(commodities)  ;
EQUATIONS     PDemand(commodities)
              PSupply(commodities)
              Equilibrium(commodities)  ;
alias (cc,commodities);
Pdemand(commodities)..
    P(commodities)=g=
       intercepts("demand",commodities)
       +sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
    intercepts("supply",commodities)
   +sum(cc,slopes("supply",commodities,cc)* Qs(cc))
    =g= P(commodities);
Equilibrium(commodities)..
     Qs(commodities)=g=  Qd(commodities);
MODEL PROBLEM /Pdemand.Qd, Psupply.Qs,Equilibrium.P/;
SOLVE PROBLEM USING MCP;
```

2.6 Dissecting the algebraic model

Sets
Data entry
Output differences

2.6.1 Sets

Above we used the subscripts i , j, commodities and cc for addressing the variable, equation and data items. In GAMS subscripts are SETs. In order to use any subscript one must declare an equivalent set.

The set declaration contains

the set name
a list of elements in the set (up to 63 characters long spaces etc allowed in quotes)

© 2022 Prof. Bruce McCarl

optional labels describing the whole set
optional labels defining individual set elements

The general format for a set statement is:

```
SET setname   optional defining text
       /      firstsetelementname    optional defining text
              secondsetelementname   optional defining text
                ...  /;
```

Examples:

(sets.gms)

```
SETs        j         /x1,x2,x3/
            i         /r1 ,r2/;
SET  PROCESS   PRODUCTION PROCESSES   /X1,X2,X3/;
SET  Commodities Crop commodities   /
            corn     in bushels,
            wheat    in metric tons,
            milk     in hundred pounds/   ;
```

More on sets appears in the Sets chapter.

2.6.1.1 Alias

One device used in the economic equilibrium formulation is the so called alias command that allows us to have a second name for the same set allowing us, in that case, to consider both the effects of own and cross commodity quantity on the demand and supply price for an item. Then general form of an Alias is

```
ALIAS(knownset,newset1,newset2,...);
```

where each of the new sets will refer to the same elements as in the existing knownset.

The command can deal with single and multidimensional set.

More on alias appears in the Sets chapter.

2.6.2 Data entry

GAMS provides for three forms of data entry. These involve PARAMETER, SCALAR and TABLE formats. Scalar entry is for scalars, Parameter generally for vectors and Table for matrices. Above I needed data for vectors and matrices but not a scalar. Nevertheless I will cover all three forms.

Scalars
Parameters

© 2022 Prof. Bruce McCarl

Tables

2.6.2.1 Scalars

SCALAR format is used to enter items that are not defined with respect to sets.

```
scalar  item1name  optional labeling text  /numerical value/
        item2name  optional labeling text  /numerical value/
            ...                           ;
```

Examples include

```
scalar     dataitem    /100/;
scalar     landonfarm total arable acres /100/;
scalars    landonfarm  /100/
           pricecorn  1992 corn price per bushel      /2.20/;
```

Scalars are covered in more depth in the Data Entry chapter.

2.6.2.2 Parameters

Parameter format is used to enter items defined with respect to sets. Generally parameter format is used with data items that are one-dimensional (vectors) although multidimensional cases can be entered. The general format for parameter entry is:

```
Parameter   itemname(setdependency)  optional text
         / firstsetelementname  associated value,
           secondsetelementname associated value,
                              ...         /;
```

Examples:

```
PARAMETER          c(j)      / x1     3    ,x2   2 ,x3    0.5/;
Parameter          b(i)      /r1 10 ,r2 3/;
PARAMETERS
         PRICE(PROCESS)     PRODUCT PRICES BY PROCESS
                                    /X1 3,X2 2,X3 0.5/;
           RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
                                    /CONSTRAIN1 10 ,CONSTRAIN2 3/;
Parameter   multidim(i,j,k) three dimensional
                              /i1.j1.k1 100 ,i2.j1.k2 90 /;
```

Notes:

- The set elements referenced must appear in the defining set. Thus when data are entered for c(j) the element names within the / designators must be in the set j.
- More than one named item is definable under a single parameter statement with a semicolon terminating the total statement.

© 2022 Prof. Bruce McCarl

- Note GAMS commands are always ended with a ; but can be multiline in nature.
- Items can be defined over up to 20 sets with each numerical entry associated with a specific simultaneous collection of set elements for each of the named sets. When multi set dependent named items are entered then the notation is

  set1elementname▪set2elementname▪set3elementname etc with periods(▪) setting off the element names in the associated sets.
- All elements that are not given explicit values are implicitly assigned with a value of zero.
- Parameters are an all-encompassing data class in GAMS into which data are kept including data entered as Scalars and Table.
- More on parameters appears in the Data Entry chapter.

2.6.2.3 Tables

TABLE format is used to enter items that are dependent on two more sets. The general format is

```
Table itemname(setone, settwo ... ) descriptive text
                   set_2_element_1   set_2_element_2
set_1_element_1      value_11          value_12
set_1_element_2      value_21          value_22;
```

Examples:

```
TABLE a(i,j) crop data
            corn  wheat cotton
    land      1     1      1
    labor     6     4      8      ;
Table intercepts(curvetype,commodities)
                   corn   wheat
         demand     4       8
         supply     1       2;
table slopes(curvetype,commodities,commodities)
                    corn  wheat
     demand.corn    -.3    -.1
     demand.wheat   -.07   -.4
     supply.corn     .5     .1
     supply.wheat    .1     .3     ;
```

Notes:

- Alignment is important. Each numerical entry must occur somewhere below one and only one column name in the Table.
- All elements that are not given explicit values or have blanks under them are implicitly assigned to equal zero.
- Items in tables must be defined with respect to at least 2 sets and can be defined over up to 20 sets. When more than two dimensional items are entered, as in the equilibrium example, periods

© 2022 Prof. Bruce McCarl

(▪) set off the element names set1elementname▪set2elementname▪set3elementname etc.

- Tables are a specific input entry format for the general GAMS parameter class of items that also encompasses scalars.
- More on tables appears in the Data Entry chapter.

2.6.2.4 Direct assignment

Data may also be entered through replacement or assignment statements. Such statements involve the use of a statement like

```
parametername(setdependency) = expression;
```

where the parameters on the left hand side must have been previously defined in a set, parameter or table statement.

Examples:

(caldata.gms)

```
scalar  a1;
scalars a2 /11/;
parameter   cc(j) , bc(j) /j2 22/;
a1=10;
a2=5;
cc(j)=bc(j)+10;
cc("j1")=1;
```

Notes:

- When a statement like cc(j)=bc(j)+10; is executed this is done for all elements in j so if j had 100,000 elements this would define values for each and every one.
- These assignments can be the sole entry of a data item or may redefine items.
- If an item is redefined then it has the new value from then on and does not retain the original data.
- The example cc("j1")=1; shows how one addresses a single specific element not the whole set, namely one puts the entry in quotes (single or double). This is further discussed in the Sets chapter.
- Calculations do not have to cover all set element cases of the parameters involved (through partial set references as discussed in the Sets chapter). Set elements that are not computed over retain their original values if defined or a zero if never defined by entry or previous calculation.
- A lot more on calculations appears in the Calculating Items chapter.

2.6.2.4.1 Algebraic nature of variable and equation specifications

When one moves to algebraic modeling the variable and equation declarations can have an added element of set dependency as illustrated in our examples and reproduced below

© 2022 Prof. Bruce McCarl

```
POSITIVE VARIABLES      x(j) ;
VARIABLES               PROFIT ;
EQUATIONS               OBJective ,
                        constraint(i) ;

POSITIVE VARIABLES      P(commodities)
                 Qd(commodities)
                 Qs(commodities) ;
EQUATIONS    PDemand(commodities)
             PSupply(commodities)
             Equilibrium(commodities)  ;
```

Such definitions indicate that these variables and equations are potentially defined for every element of the defining set (also called the domain) thus x could exist for each and every element in j. However the actual definition of variables does not occur until the .. equation specifications are evaluated as discussed next. More on set dependent variable and equation definitions appears in the Variables, Equations, Models and Solves chapter.

2.6.2.4.2 Algebra and model .. specifications

The equations and variables in a model are defined by the evaluation of the .. equation specifications. The .. equations for our examples are

```
OBJective.. PROFIT=E=   SUM(J,c(J)*x(J)) ;
constraint(i).. SUM(J,a(i,J) *x(J))  =L= b(i);

Pdemand(commodities)..
    P(commodities)=g=
       intercepts("demand",commodities)
       +sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
    intercepts("supply",commodities)
   +sum(cc,slopes("supply",commodities,cc)* Qs(cc))
    =g= P(commodities);
Equilibrium(commodities)..
     Qs(commodities)=g=  Qd(commodities);
```

Here GAMS will operate over all the elements in the sets in each term. For example, in the OBJective equation GAMS will add up the term c(J)*x(J) for all set elements in j. Similarly, the equation constraint (i) will define a separate constraint equation case for each element of i. Also within the equation case associated with an element of i only the elements of a(i,j) associated with that particular i will be included in the term SUM(J,a(i,J) *x(J)). Similarly, within the second example equations of each type are included for each member of set commodities.

Notes:

- These examples show us moving away from the data specification that we were employing in the

© 2022 Prof. Bruce McCarl

GAMS the early GAMS examples in this chapter. In particular rather than entering numbers in the model we are now entering data item names and associated set dependency. This permits us to specify a model in a more generic fashion as will be discussed in a later section of this tutorial on virtues of algebraic modeling.

- The only variables that will be defined for a model are those that appear with nonzero coefficient somewhere in at least one of the equations defined by the .. equations.
- More on .. specifications appears within the Variables, Equations, Models and Solves chapter.

2.6.3 Output differences

When set dependency is used in association with variables and equations and model then this changes the character of a few of the output items. In particular, there are some changes in the equation listing, variable listing, and solution reports for variables and equations.

Equation listing
Variable list
Equation solution report
Variable solution report

2.6.3.1 Equation listing

The equation listing exhibits a few different characteristics in the face of set dependent variable and equation declarations. In particular, the variables declared over sets are reported with a display of their set dependency encased in parentheses. Also the equations declared over sets have multiple cases listed under a particular equation name. An example is presented below in the context of our core optimization example (**optimize.gms**) and shows three cases of the x variable (those associated with the corn, wheat, and cotton set elements). It also shows that two cases are present for the equation called constraint (land and labor).

```
---- OBJective  =E=
OBJective..  - 109*x(Corn) - 90*x(Wheat) - 115*x(Cotton) + PROFIT =E= 0 ; (LHS =

---- constraint  =L=
constraint(Land)..  x(Corn) + x(Wheat) + x(Cotton) =L= 100 ; (LHS = 0)
constraint(Labor)..  6*x(Corn) + 4*x(Wheat) + 8*x(Cotton) =L= 500 ; (LHS = 0)
```

A portion of the equation listing from a more involved example (**model.gms**) also reveals additional differences. In the TCOSTEQ equation that we see the portrayal of coefficients involved with several declared variables: 3 cases of Build, 6 cases of Shipsw, 6 cases of Shipwm and 4 cases of Shipsm. The **model.gms** example also shows what happens there are more cases of equation than the number of equation output items output by default as controlled by the option Limrow (as discussed in the Standard Output chapter). In this case Limrow was set to 2 but there were three cases of the equation named Capacity and GAMS indicates that one case was skipped. If there had been 100, then 98 would have been skipped.

```
---- TCOSTEQ  =E=  TOTAL COST ACCOUNTING EQUATION
TCOSTEQ..  Tcost - 50*Build(A) - 60*Build(B) - 68*Build(C) - Shipsw(S1,A) - 2*Sh
      - 8*Shipsw(S1,C) - 6*Shipsw(S2,A) - 3*Shipsw(S2,B) - Shipsw(S2,C) - 4*Ship
```

© 2022 Prof. Bruce McCarl

```
      - 6*Shipwm(A,D2) - 3*Shipwm(B,D1) - 4*Shipwm(B,D2) - 5*Shipwm(C,D1) - 3*Sh
      - 4*Shipsm(S1,D1) - 8*Shipsm(S1,D2) - 7*Shipsm(S2,D1) - 6*Shipsm(S2,D2) =L
      (LHS = -4, INFES = 4 ***)

---- CAPACITY  =L=  WAREHOUSE CAPACITY
CAPACITY(A)..  - 999*Build(A) + Shipwm(A,D1) + Shipwm(A,D2) =L= 0 ; (LHS = 0)
CAPACITY(B)..  - 60*Build(B) + Shipwm(B,D1) + Shipwm(B,D2) =L= 0 ; (LHS = 0)

REMAINING ENTRY SKIPPED
```

2.6.3.2 Variable list

The variable listing also exhibits a few different characteristics in the face of set dependent variable and equation declarations. In particular, the variables declared over sets have multiple cases listed under a particular variable name as do any involved sets. An example is presented below in the context of our core optimization example (**optimize.gms**) and shows three cases of the x variable (those associated with the corn, wheat, and cotton set elements). It also shows that the variables use resources from two cases of the equation called constraint (land and labor).

```
---- x
x(Corn)
                (.LO, .L, .UP = 0, 0, +INF)
     -109       OBJective
        1       constraint(Land)
        6       constraint(Labor)
x(Wheat)
                (.LO, .L, .UP = 0, 0, +INF)
      -90       OBJective
        1       constraint(Land)
        4       constraint(Labor)
x(Cotton)
                (.LO, .L, .UP = 0, 0, +INF)
     -115       OBJective
        1       constraint(Land)
        8       constraint(Labor)
```

A portion of the variable listing from the more involved **model.gms** example shows GAMS indicating four cases were skipped when Limcol was smaller than the number of cases on hand (as discussed in the Standard Output chapter).

```
---- Shipsw  Amount Shipped To Warehouse
Shipsw(S1,A)
                (.LO, .L, .UP = 0, 0, 1000)
       -1       TCOSTEQ
        1       SUPPLYEQ(S1)
       -1       BALANCE(A)
Shipsw(S1,B)
                (.LO, .L, .UP = 0, 0, 1000)
       -2       TCOSTEQ
```

© 2022 Prof. Bruce McCarl

```
     1      SUPPLYEQ(S1)
    -1      BALANCE(B)

REMAINING 4 ENTRIES SKIPPED
```

2.6.3.3 Equation solution report

The equation solution LST also shows all existing cases grouped under each equation name when set dependency is present as illustrated below in the context of our core optimization example (**optimize.gms**).

```
---- EQU constraint
            LOWER      LEVEL      UPPER    MARGINAL
Land         -INF    100.000    100.000     52.000
Labor        -INF    500.000    500.000      9.500
```

2.6.3.4 Variable solution report

The variable solution LST segment also shows all existing cases grouped under each variable name when set dependency is present as illustrated below in the context of our core optimization example (**optalgebra.gms**).

```
---- VAR x
            LOWER      LEVEL      UPPER    MARGINAL
Corn          .        50.000      +INF        .
Wheat         .        50.000      +INF        .
Cotton        .           .        +INF     -13.000
```

2.7 Good modeling practices

Above I have covered the essential GAMS features one would employ in any modeling exercise. However I have not done very good job of exploiting a major GAMS capability involved self-documentation. In any modeling exercise there are an infinite variety of choices that can be made in naming the variables, equations, parameters, sets etc. and formatting their presentation in the GMS instruction file. Across these choices that can be large differences in the degree of self-documentation within the GMS code. In particular, as explained in the chapter on Rules for Item Names, Element names and Explanatory Text, one employs short names like x(j) as in **optalgebra.gms** or longer names (up to 63 characters) for the variables like production(products). I advocate use of longer names to enhance the readability of the document.

The GAMS also permits one to add comments, for example telling what is being done by particular instructions or indicating data sources. This can be done by a number of means including typing lines beginning with an * in column one or encasing longer comments between a $ONTEXT and $OFFTEXT. GAMS elements for including comments are discussed in the chapter entitled Including Comments.

I illustrate the longer name and comment capability along with improved spacing and line formatting in the context of the model **optalgebra.gms** creating the new model **goodoptalgebra.gms**. The two models use the same data and get the same answer only the item names and formatting have been changed. In my judgment, the longer names substantially contribute to self-documentation and make it easier to go back to use a model at a future time or transfer a model to others for their use. More material on the formatting subject appears in the Writing Models and Good Modeling Practices chapter.

© 2022 Prof. Bruce McCarl

Original version

(optalgebra.gms)

```
SET     j               /Corn,Wheat,Cotton/
        i               /Land ,Labor/;
PARAMETER
  c(j)      / corn     109    ,wheat   90 ,cotton    115/
  b(i)     /land 100 ,labor 500/;
TABLE a(i,j)
              corn    wheat  cotton
  land          1       1       1
  labor         6       4       8      ;
POSITIVE VARIABLES   x(j);
VARIABLES            PROFIT             ;
EQUATIONS            OBJective          ,  constraint(i) ;
 OBJective.. PROFIT=E=   SUM(J,(c(J))*x(J)) ;
 constraint(i).. SUM(J,a(i,J) *x(J))  =L= b(i);
MODEL RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

Revised version with comments in blue

(goodoptalgebra.gms)

```
*well formatted algebraic version of model optalgebra.gms
SET       Products  Items produced by firm
              /Corn   in acres,
               Wheat  in acres ,
               Cotton in acres/
          Resources  Resources limiting firm production
              /Land   in acres,
               Labor  in hours/;
PARAMETER Netreturns(products)  Net returns per unit produced
              /corn 109 ,wheat 90 ,cotton 115/
          Endowments(resources) Amount of each resource available
              /land 100 ,labor 500/;
TABLE     Resourceusage(resources,products) Resource usage per unit produced
                          corn    wheat  cotton
               land          1       1       1
               labor         6       4       8      ;
POSITIVE VARIABLES   Production(products) Number of units produced;
VARIABLES            Profit               Total fir summed net returns ;
EQUATIONS            ProfitAcct           Profit accounting equation ,
                     Available(Resources) Resource availability limit;
$ontext
      specify definition of profit
$offtext
 ProfitAcct..
      PROFIT
      =E= SUM(products,netreturns(products)*production(products)) ;
```

© 2022 Prof. Bruce McCarl

```
$ontext
      Limit available resources
      Fix at exogenous levels
$offtext
 available(resources)..
      SUM(products,
          resourceusage(resources,products) *production(products))
      =L= endowments(resources);

MODEL RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

2.8 Structure of GAMS statements, programs and the ;

Now that I have been through the most essential basic elements of the GAMS syntax, I can review the general format of GAMS statements and GMS files. A GAMS program is a collection of statements in the GAMS language. A number of comments can be made about how the file needs to be formatted

- Statements must be ordered so that items are initially declared before they are used. If they are used on the right hand side of a calculation (an = statement) they also must be given data before use. If they are used in a model equation then they must be given data before a Solve appears. This is enforced by GAMS indicating a lack of declaration and numerical specification as a compilation error so one does not need to meticulously check order of declaration, definition and use.[1]
- Individual GAMS statements can be formatted in almost any style. Multiple lines may be used for a statement, blank lines can be embedded, any number of spaces or tabs may be inserted and multiple statements may be put on one line separated by a ;
- Every GAMS statement should be terminated with a semicolon, as all the examples in this book illustrate.
- GAMS is not case sensitive, thus it is **equivalent** to type the command VARIABLE as variable or the variable names XCOTTON as XcOttoN. However, there is case sensitivity with respect to the way things are printed out with the first presentation being the one used as discussed in the Rules for Item Capitalization and Ordering chapter.
- The use of a named item (which in GAMS can be a set, parameter, scalar, table, acronym, variable, equation, model or file) involves three steps:

  Declaration where one announces the existence of a named item giving it a name.
  Assignment giving it a specific value or replacing its value with the results of an expression.
  Subsequent usage.
- The item names, elements and explanatory text must follow certain rules as discussed in the Rules for Item Names, Element names and Explanatory Text chapter.

[1] **This and a number of the other points in this section are adapted from Richard E. Rosenthal's "A GAMS Tutorial" that appeared in the GAMS Users Guide documents by Brooke et al.**

© 2022 Prof. Bruce McCarl

2.9 Adding complexity

There are a few more topics meritorious of coverage in this tutorial that involve GAMS capabilities to include conditionals, display data, do calculations incorporating optimal solution information and solve a model more than once. Each is discussed below.

Conditionals
Displaying data
Report writing

2.9.1 Conditionals

Certainly when doing calculations and setting up models cases arise where one might wish to do different things conditional upon data. In particular, one might wish to do a calculation like z=x/y only if y is nonzero or one might wish to define demand equations only for cases where demand exists. Incorporation of such considerations into GAMS program involves what's known as the $conditional as extensively discussed in the Conditionals chapter. Below I present several examples of this feature. Generally the expressions are of the form

```
term$logical condition
```

which says do something with term only if the logical condition is true where the $ can be read as if it were the word if. Conditionals can appear in a number of contexts, as I will illustrate below.

2.9.1.1 Conditionally execute an assignment

The condition

```
X$(y gt 0) = 10;
```

says set X=10 if the scalar y is greater than zero, while the condition

```
percentchange$(y ne 0)= 100*(x-y)/y;
```

says compute the item percentchange if y is not equal to zero.

For more on this class of conditionals see the discussion in the Conditionals chapter.

2.9.1.2 Conditionally add a term in sum or other set operation

The condition

```
z=sum(i$(y(i) gt 0),x(i));;
```

says include the term for set element i only if y(i) > 0, while

© 2022 Prof. Bruce McCarl

```
z=sum((i,j)$(sameas(i,j)),x(i,j));
```

says add the term corresponding to a pair of set elements i and j only if the set elements have the same name (thus if the name of element i was Chicago then the j term would be included in the sum only if the name of element j was Chicago).

For more on this class of conditionals see the discussion in the Conditionals chapter. For more on Sameas also see the Conditionals chapter.

2.9.1.3 Conditionally define an equation

The conditions

```
Eq1$(qq gt 0)..    xvar=e=3;
Eq2$(sum(I,q(i)) gt 0)..  yvar=l=4;
Eq3(i)$(a(i) gt 0)..   ivar(i)=g= -a(i);
```

each cause an equation to exist in a model only if the condition is satisfied.

For more on this class of conditionals see the discussion in the Conditionals chapter.

2.9.1.4 Conditionally include a term in an equation

The conditions

```
Eq4 . .    xvar+yvar$(qq gt 0)=e=3;
X=sum(I,q(i))$(qq gt 0)+4;
Q(i)=a(i)+1$(a(i) gt 0);
```

each cause the term in red to only be included in an expression (it is treated as zero otherwise) only if the condition is satisfied.

For more on this class of conditionals see the discussion in the Conditionals chapter.

2.9.2 Displaying data

One may display any GAMS parameter, set, variable attribute, equation attribute or model attribute as well as quoted text using the GAMS display statement. Generally the display is of the format

```
DISPLAY ITEM1,ITEM2,ITEM3;
```

where the items are either

- Quoted strings in single or double quotes such as

```
Display 'here it is', "hello";
```

- Parameter or set names without any referencing to setdependency. Thus in **dispord.gms** while the parameter data is defined over 4 sets

© 2022 Prof. Bruce McCarl

```
parameter data(index1,index2,index3,index4);
```

I simply say

```
display data;
```

- Variable, equation or model attributes with the item name and attribute desired specified

```
Display x.l, eq.m;
```

- Multiple items can be listed in a display statement separated by commas.

Notes:

- Display will not print out items that are zero leaving blanks or skipping items where entire rows or columns are zero.
- GAMS displays can be enhanced in terms of form, and content in several ways as discussed in the Improving Output via Report Writing chapter. One way involves use of an option command of the following form

```
OPTION ITEMNAME:DECIMAL:ROWitems:COLUMNitems
```

which will cause all subsequent displays of the named item to follow rules specified by three numbers following the colons which are

```
DECIMAL     number of decimal places to be included
ROWitems    number of indices displayed within rows
COLUMNitems number of indices displayed within columns
```

A display formatting sequence is introduced into the optimization example (**goodoptalgebra.gms**) as follows:

```
option thisreport:2:1:2;
display thisreport;
```

which says use 2 decimal places and produce a display with 1 item in the rows and 2 in the columns yielding

```
            Total      Use by      Use by    Marginal
        Available         Corn       Wheat       Value

Land       100.00       50.00       50.00       52.00
Labor      500.00      300.00      200.00        9.50
```

A display of the same item with option this report:4:2:1; yields

```
                      Corn       Wheat   Available       Value

Land .Total                                 100.0000
Land .Use by       50.0000     50.0000
Land .Marginal                                          52.0000
```

© 2022 Prof. Bruce McCarl

```
Labor.Total                              500.0000
Labor.Use by      300.0000    200.0000
Labor.Marginal                                          9.5000
```

2.9.3 Report writing

GAMS permits one to do calculations using solution information to improve the information content of the output. This exercise is commonly called report writing. Information relative to the variable, equation and model solution is passed to GAMS from solvers. These data can be used in report writing computations.

In GAMS the solution level for a variable is Variablename.L while it is Equationname.L for an equation. The dual or shadow price information for an equation is addressed as Equationname.M and the reduced cost for a variable is Equationname.M. The numerical values of these parameters are generally undefined until a solve is performed and retains the value from the most recent solve from then on. In the algebraic version of the equilibrium model (**econequilalg.gms**) I introduce the following report writing sequence

```
set qitem /Demand, Supply, "Market Clearing"/;
set item /Quantity,Price/
parameter myreport(qitem,item,commodities);
myreport("Demand","Quantity",commodities)= Qd.l(commodities);
myreport("Supply","Quantity",commodities)= Qs.l(commodities);
myreport("Market Clearing","Price",commodities)= p.l(commodities);
display myreport;
```

which saves the supply and demand quantities along with the market clearing price. The resultant report is generated with a display statement and is

```
----     39 PARAMETER myreport

                                  Corn        Wheat

Supply          .Quantity        1.711        8.156
Demand          .Quantity        1.711        8.156
Market Clearing.Price            2.671        4.618
```

where I have color coded the originating statements and resultant output.

A report writing sequence is also introduced into the optimization example (**goodoptalgebra.gms**) as follows

```
set item  /Total,"Use by",Marginal/;
set qitem /Available,Corn,Wheat,Cotton,Value/;
parameter Thisreport(resources,item,qitem) Report on resources;
Thisreport(resources,"Total","Available")=endowments(resources);
Thisreport(resources,"Use by",qitem)=
    sum(products$sameas(products,qitem),
        resourceusage(resources,products) *production.l(products));
Thisreport(resources,"Marginal","Value")=
         available.m(resources);
```

© 2022 Prof. Bruce McCarl

```
option thisreport:2:1:2;
display thisreport;
```

where both equation marginals (shadow prices) and variable levels are included in the report writing calculations. This yields the report

```
            Total      Use by      Use by    Marginal
        Available       Corn       Wheat       Value

Land       100.00      50.00      50.00       52.00
Labor      500.00     300.00     200.00        9.50
```

where I have color coded the originating statements and resultant output.

The report wring topic is extensively discussed in the Improving Output via Report Writing chapter with a more advanced discussion also appearing in the Output via Put Commands chapter.

2.10 Why use GAMS and algebraic modeling

Finally I feel it is beneficial to examine the attributes and difficulties with GAMS based algebraic modeling. This is done under the following topics

Use of algebraic modeling
- Context changes
- Expandability
- Augmentation

Aid with initial formulation and subsequent changes
Adding report writing
Self-documenting nature
Large model facilities
Automated problem handling and portability
Model library and widespread professional use
Use by Others
Ease of use with Non Linear, Mixed Integer, CGE and other problem forms
Interface with other packages

2.10.1 Use of algebraic modeling

GAMS permits one to express a formulation in general algebraic terms using symbolic summation notation. This allows modelers to concisely state problems, largely independent of the data and exact application context. Such formulations are inherently expandable, easily subjected to context changes, and easily augmented as will be discussed just below.

However, use of algebraic modeling can be a two edged sword. GAMS algebraic requirements and summation notation are difficult for some users. Some people will always desire to deal with the exact problem context, not an abstract general formulation. This does lead to a strategy most modelers use when employing GAMS modeling. Namely, GAMS exercises are usually supported by small hand formulations that capture problem essence and serve as an aid in GAMS model formulation.

© 2022 Prof. Bruce McCarl

2.10.1.1 Context changes

Consider the optimization example from above (**goodoptalgebra.gms**) which involved a farming example. This can be rewritten to another context as follows (**newcontext.gms**)

```
SET       Products  Items produced by firm
              /Chairs  , Tables , Dressers /
          Resources  Resources limiting firm production
              /RawWood , Labor  , WarehouseSpace/;
PARAMETER Netreturns(products)  Net returns per unit produced
              /Chairs 19  , Tables 50, Dressers 75/
          Endowments(resources) Amount of each resource available
              /RawWood 700 , Labor 1000 , WarehouseSpace 240/;
TABLE     Resourceusage(resources,products) Resource usage per unit produced
                          Chairs   Tables  Dressers
          RawWood            8        20      32
          Labor             12        32      45
          WarehouseSpace     4        12      10   ;
POSITIVE VARIABLES   Production(products) Number of units produced;
VARIABLES            Profit               Total fir summed net returns ;
EQUATIONS            ProfitAcct           Profit accounting equation ,
                     Available(Resources) Resource availability limit;
 ProfitAcct..
      PROFIT
      =E= SUM(products,netreturns(products)*production(products)) ;
 available(resources)..
      SUM(products,
          resourceusage(resources,products) *production(products))
      =L= endowments(resources);
MODEL RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

where only the lines in black changed not those in red relative to the farming example. So what? The algebraic structure once built did not need to be altered and GAMS models can easily be changed from one context to another.

2.10.1.2 Expandability

Consider the **newcontext.gms** optimization example from just above. That examples depicts production of three products from three resources. One could add two new products and two new resources as follows (**expand.gms**)

```
SET       Products  Items produced by firm
              /Chairs  , Tables , Dressers, HeadBoards, Cabinets /
          Resources  Resources limiting firm production
              /RawWood , Labor  , WarehouseSpace , Hardware, ShopTime/;
PARAMETER Netreturns(products)  Net returns per unit produced
            /Chairs 19,Tables 50,Dressers 75,HeadBoards 28,Cabinets 25/
          Endowments(resources) Amount of each resource available
```

© 2022 Prof. Bruce McCarl

```
            /RawWood 700,Labor 1000,WarehouseSpace 240,Hardware 100, Shoptime 60
TABLE     Resourceusage(resources,products) Resource usage per unit produced
                          Chairs   Tables  Dressers HeadBoards Cabinets
          RawWood            8        20      32         22      15
          Labor             12        32      45         12      18
          WarehouseSpace     4        12      10          3       7
          Hardware           1         1       3          0       2
          Shoptime           6         8      30          5      12;
POSITIVE VARIABLES   Production(products) Number of units produced;
VARIABLES            Profit               Total fir summed net returns ;
EQUATIONS            ProfitAcct           Profit accounting equation ,
                     Available(Resources) Resource availability limit;
 ProfitAcct..
      PROFIT
      =E= SUM(products,netreturns(products)*production(products)) ;
 available(resources)..
      SUM(products,
          resourceusage(resources,products) *production(products))
      =L= endowments(resources);
MODEL RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

where only the material in black was added with no alterations of that in red relative to the **newcontext.gms** example. So what? The algebraic structure once built did not need to be altered and GAMS models can easily be expanded from smaller to larger data sets. Such capabilities constitute a major GAMS model development strategy. One can originally develop a model with a small data set and fully debug it. Then later one can move to the full problem data set without having to alter any of the algebraic structure but have confidence in the algebraic structure. This is discussed further in the Small to Large: Aid in Development and Debugging chapter.

2.10.1.3 Augmentation

Consider the **newcontext.gms** optimization example from just and suppose we wish to augment the model with constraints and variables reflecting the capability to rent or hire additional resources subject to a maximum availability constraint. This is done in the following example (**augment.gms**)

```
SET       Products  Items produced by firm
              /Chairs  , Tables , Dressers /
          Resources  Resources limiting firm production
              /RawWood , Labor  , WarehouseSpace/
          Hireterms  Resource hiring terms
              /Cost , Maxavailable /;
PARAMETER Netreturns(products)  Net returns per unit produced
              /Chairs 19  , Tables 50, Dressers 75/
          Endowments(resources) Amount of each resource available
              /RawWood 700 , Labor 1000 , WarehouseSpace 240/;
TABLE     Resourceusage(resources,products) Resource usage per unit produced
                          Chairs   Tables  Dressers
          RawWood            8        20      32
          Labor             12        32      45
```

© 2022 Prof. Bruce McCarl

```
          WarehouseSpace     4        12      10   ;
Table     Hiredata(Resources,hireterms)  Resource hiring data
                          Cost   Maxavailable
          RawWood            3        200
          Labor             12        120
          WarehouseSpace     4        112;
POSITIVE VARIABLES   Production(products)    Number of units produced
                     HireResource(Resources) Resources hired;
VARIABLES            Profit                  Total firm summed net returns ;
EQUATIONS            ProfitAcct              Profit accounting equation ,
                     Available(Resources)    Resource availability limit
                     Hirelimit(Resources)    Resource hiring limit;
 ProfitAcct..
      PROFIT
      =E= SUM(products,netreturns(products)*production(products))
         -SUM(resources,hiredata(resources,"cost")* HireResource(Resources))
 available(resources)..
      SUM(products,
          resourceusage(resources,products) *production(products))
      =L= endowments(resources) +  HireResource(Resources);
Hirelimit(Resources)..
          HireResource(Resources) =l= hiredata(resources,"maxavailable");
MODEL RESALLOC /ALL/;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

where only the material in black was added with no alterations of that in red relative to the **newcontext.gms** example. So what? The algebraic structure from the other study could be used supplied the core of the new model with structural features added as needed. Such a capability constitutes another major GAMS model development strategy.

One can adapt models from other studies customizing them for the problem at hand speeding up the development process. In addition to adapting models from related studies done by the modeler or the group in which the modeler works, there are number of other sources one may be able to exploit to jumpstart a model development project. This is further discussed **below**.

2.10.2 Aid with initial formulation and subsequent changes

GAMS aids both in initially formulating and subsequently revising formulations. GAMS facilitates specification and debugging of an initial formulation by allowing the modeler to begin with a small data set, then after verifying correctness expand to a much broader context. For example, one could initially specify a small transportation model with a few suppliers and demanders. Then after that model is debugged one could expand the problem to encompass fifty shipping origins and two hundred destinations without needing to change the algebraic model as discussed in the Small to Large: Aide in Development and Debugging chapter and the expandability section above.

GAMS also makes it easy to alter the model. Large models in programs like spreadsheets can be difficult to modify. In a spreadsheet, I find it hard to add in a set of new constraints and variables properly interjecting all the linkages and cannot figure out how to easily get a model right with a few commodities then automatically expand the model scope to many commodities and locations as illustrated in the expandability section above. On the other hand, GAMS allows one to add model

© 2022 Prof. Bruce McCarl

features much more simply. Generally, modelers do not try to make a complete formulation the first time around. Rather one starts with a small formulation and then adds structural features as needed adding features as illustrated in the augmentation section above. GAMS also enforces consistent modeling, allowing models to be transferred between problem contexts as shown above.

2.10.3 Adding report writing

Generally, default GAMS output for the model solution is not adequate for conveying solution information to the modeler or associated decision-makers. One often does calculations using solution information to improve information content of the GAMS output. This is elaborated upon in the Improving Output via Report Writing chapter below.

2.10.4 Self-documenting nature

One important GAMS feature its self-documenting nature. Modelers can use long variable, equation and index names as well as comments, data definitions etc., allowing a readable and fairly well documented problem description. Model structure, assumptions, and any calculation procedures used in the report writing are documented as a byproduct of the modeling exercise in a self-contained file. Comment statements can be inserted by placing an asterisk in column one, followed by text identifying data sources or particular assumptions being used (i.e., in some of the my models, comments identify data source publication and page). Under such circumstances GAMS allows either the original author or others to alter the model structure and update data.

Consider for example the following example. Can you figure out what context the example is from?

```
LABOR(Farm)..
    PLOWLAB(Farm) * PLOW(Farm)
      + SUM( crop, PLNTLAB(Farm,Crop) *PLANT(Farm,Crop)
  + HARVLAB(Farm,Crop) * HARVEST(Farm,Crop) )
      =L=  LABORAVAIL(Farm);
```

2.10.5 Large model facilities

GAMS is not the tool of choice for small, infrequently solved problems. In such cases, the generality of the presentation may not be worth the effort, and spreadsheet or other formulations are probably quicker and easier to deal with. GAMS is best employed for medium or large sized models (more than 100 rows and/or columns) and can handle large problems as the table of a few or my application model sizes below indicates.

MODELS	VARIABLES	EQUATIONS	NOTES ON IMPLEMENTATION
10 REGION ASM	9860	811	412 crop budgets 129 livestock 45423 lines 2.9Mb
ASM	30146	2844	1662 crop budgets

© 2022 Prof. Bruce McCarl

			838 livestock budgets 60469 lines 8.3Mb
SOIL ASM	41574	2935	123087 lines 33.6Mb
GLOBAL ASM(sto)	305605	14556	120991 lines 43.5Mb
FASOM	26012	1774	141697 lines 35.3Mb
HUMUS	429364	236234	41444 lines 123.1Mb
EDWARD	12161	5655	7858 lines 6.1Mb

The gains to using GAMS rise with problem size and complexity of the model use exercise study. When a modeler deals with large problems, the GAMS algebraic statement is probably the only thing that is thoroughly understood. Often the numerical formulation has grown out of control.

2.10.6 Automated problem handling and portability

Many of the tasks that would traditionally have required a computer programmer are automated. As such, GAMS automatically does coefficient calculation; checks the formulation for obvious flaws; chooses the solver; formats the programming problem to meet the exact requirements of the solver; causes the solver to execute the job; saves and submits the advanced basis when doing related solutions; and permits usage of the solution for report writing. Also GAMS verifies the correctness of the algebraic model statements and allows empirical verification using programs like GAMSCHK.

Furthermore, GAMS code is portable between computers. GAMS has been implemented on machines ranging from PCs to UNIX/LINUX workstations to CRAY super computers. Exactly the same code runs on all of these computer systems.

Switching solvers is simple, requiring changing a solver option statement, or changing from using LP to using NLP, as discussed in the Variables, Equations, Models and Solves chapter. Links to spreadsheets have also been developed as discussed in the Links to Other Programs Including Spreadsheets chapter.

2.10.7 Model library and widespread professional use

Today GAMS has become the de facto standard for optimization modeling in many fields. Modelers may be able to adapt models or gain insights from others. Some sources of models from which model features can be adapted include:

- Models from experienced users that address similar problems that are closely related in concept or structure and can be adapted.
- Models associated with textbooks. For example, my book with Spreen contains many examples. The book and the examples are available through my Web page *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/books.htm*

© 2022 Prof. Bruce McCarl

- Models are available through the GAMS library which is directly included in the IDE. These cover many different settings.
- References from the GAMS web pages *http://www.gams.com/*, or *http://gamsworld.org/*.

Each of these resources along with others are discussed in the chapter called Application Help: Model Library, Web Sites, Documentation.

2.10.8 Use by Others

Modeling personnel are often rare. For example, in international development contexts, detailed GAMS applications have been set-up by modeling experts but subsequently, the model is utilized by policy-makers with minimal, if any, assistance from the modeling experts. Often, given proper internal documentation and a few instructions, clerical labor and nontechnical problem analysts can handle an analysis.

2.10.9 Ease of use with NLP, MIP, CGE and other problem forms

GAMS handles a variety of different problem types and has become one of principal languages for computable general equilibrium modeling, agricultural economic modeling and oil refinery modeling. It is also one of the principal platforms for experimentation with developing fields like mixed integer nonlinear programming models and global optimization models. GAMS Corporation is continually engaged in an effort to provide the most recent available solver software. This likely implies that GAMS users will have available the most recent developments in solver software and libraries of application test problems in emerging fields.

2.10.10 Interface with other packages

While not as well developed as I would like, GAMS does have procedures to interface with other programs like spreadsheets, databases, custom control programs, and Visual basic procedures among others. These interfaces are discussed in the chapter entitled to Links to Other Programs Including Spreadsheets.

3 Language Basics

This section covers the real basics of the GAMS language covering how to specify a basic program. The coverage is organized by chapter with the chapters covering:

Sets
Data Entry
Variables, Equations, Models and Solves
Model Types and Solvers
Standard Output
Writing Models and Good Modeling Practices

3.1 Sets

Within GAMS, sets are equivalent to subscripts in algebra and are a series of items that can be

© 2022 Prof. Bruce McCarl

simultaneously operated over including summed over or looped over among other possibilities. Here material is given on the major aspects of sets usage under the following categories:

Set declaration
Subsets
Element definition
Multi dimensional sets
Domain checking
Set element referencing
Universal Set: * as a set identifier
Finding sets from data
Element order and capitalization in output
Functions specifically referencing sets
Indexing sets defined over time
Set Arithmetic

3.1.1 Set declaration

In order to use any set one must first declare it. In its most complete form the set declaration contains

set name	(rules for item names)
optional explanatory text for the whole set	(rules for entries)
list of elements contained surrounded by /'s	(rules for set element names)
optional explanatory text for the whole set	(rules for entries)
optional explanatory text for individual elements	(rules for entries)

The general format for the set declaration and element definition statement is:

```
SET setname optional explanatory text
 /    first set element name        Optional explanatory text
      second set element name       Optional explanatory text
        …
 /;
```

or

```
SETs setname optional explanatory text
/     first set element name        Optional explanatory text
      second set element name       Optional explanatory text
        …
 /;
```

Examples:

(sets.gms)

```
SETs     j                                  /x1,x2,x3/
         i                                  /r1 ,r2/;
```

```
SET      PROCESS     PRODUCTION PROCESSES          /X1,X2,X3/;
SET      commodities Crop commodities         / corn   in bushels
                                               Wheat  in metric tons
                                               milk   in pounds
                                               cost   "cost/unit"
                                               "long-complex-*&$name"
                                               'element name'/ ;
Set      jj(j)       set to b e computed later without entries ;
$onmulti
set      i           additional entries for i   /i1,i2/;
set      composite(i,j) multidimensional     /r1.set.j/;
set       kk             kk has all of i and j in it /set.i,set.j/;
```

Notes:

- The word set or sets can be used.
- Set names must obey the item naming presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- Multiple sets can be stacked with the set or sets keyword only used once (see example with i and j below). When multiple sets are defined in one set statement a ; is entered after all set definitions.
- Set elements are separated by spaces or commas.
- Element definitions can be quoted, have blanks or special characters as discussed in the Rules for Item Names, Element names and Explanatory Text chapter.
- Use of $onmulti allows multiple set statements to appear for a named set.
- One can add the entire contents of a previously defined set to the definition of another set using the syntax set.setname as used to inclide sets i and j in the definition of set kk above

3.1.2 Singleton Sets

GAMS permits a type of set which is called a Singleton Set. This is a special case of a GAMS Set which has at most a single element (it may also be empty with zero elements).

In its most complete form the singleton set declaration contains

```
   Singleton set
set name
                                          (rules for item names)
optional explanatory text for the whole set          (rules for entries)
the single element contained surrounded by /'s       (rules for set element
                                          names)
optional explanatory text for the individual element (rules for entries)
```

The general format for the set declaration and element definition statement is:

```
   Singleton SET setname optional explanatory text
```

© 2022 Prof. Bruce McCarl

```
       /    Only set element name         Optional explanatory text
       /;
```

or

```
Singleton SETs setname optional explanatory text
/     Only set element name         Optional explanatory text
/;
```

Examples:

(singleton.gms)

```
Singleton set s /oneelement/;
Set a /a1*a3/;
set b /b1*b3/    ;
Singleton set subsetofa(a) /a1/;
Singleton set subsetofb(b);
singleton set multidim(a,b) /a2.b3/ ;
subsetofb("b3")=yes;
parameter xx(a) /a1 10, a2 4, a3 6/;
parameter xy(b) /b1 10, b2 4, b3 6/;
scalar zz;
zz=xx(subsetofa);
display xx,subsetofa,zz;
zz=xx(subsetofa)+100*xy(subsetofb);
display zz;
positive variables myy(a)
                   qq;
variable           obj;
equations                objeq
                   limitmyy(a)
                   pegzz;
objeq..           obj=e= qq+xx(subsetofa)*myy(subsetofa);
limitmyy(a)..     myy(a)=g= xx(a);
pegzz..           qq=e=xx(subsetofa)/2;
model singletest /all/;
solve singletest minimizing obj using lp;
display xx,subsetofa,qq.l,myy.l;
Subsetofa("a1")=yes;
display Subsetofa;
Subsetofa("a2")=yes;
display Subsetofa;
```

Comments:

- In the example **s, subsetofa** and **multidim** are singleton sets in simple set, subset and tuple forms
- Singleton Sets can be used in assignment statements without being controlled by an index operator like a sum loop or family name as illustrated by the following lines from the example.

© 2022 Prof. Bruce McCarl

```
zz=xx(subsetofa);
zz=xx(subsetofa)+100*xy(subsetofb);
objeq..          obj=e= qq+xx(subsetofa)*myy(subsetofa);
pegzz..          qq=e=xx(subsetofa)/2;
```

- When assigning to a singleton set it is automatically cleared out first thus using the commands

```
Subsetofa("a1")=yes;
Subsetofa("a2")=yes;
```

  Would result in Subsetofa containing only a2 after execution.

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

```
Singleton Set s / s1*s3 /;
```

  **** $844

  Error Message

  844 Singleton with more than one entry

- The behavior of assignments to **Singleton Sets** can be influenced by the option strictSingleton or the dollar commands $onStrictSingleton/$offStrictSingleton:

3.1.3 Subsets

One can define subsets containing part of the elements of another set using a set statement. The general format is

```
SET   subsetname(setname)  optional explanatory text
                        /  Elementname1  optional explanatory text
                           Elementname2 optional explanatory text/;
```

where most of the contents are as discussed under set declaration above. The new elements are

Subsetname	which names this subset.
Setname	which names the "super" set that this one is a subset of.

Examples:

(sets.gms)

```
Set    Superset              /r1,r2,r4*r15, r25/
       Subset(superset)      /r1,r25/;
Set    allitems              /Corn,wheat,water,land/
```

© 2022 Prof. Bruce McCarl

```
Crop(allitems)          /Corn,wheat/
Resources(allitems)     /water,land/;
```

Notes:

- The named elements of the subset must be elements of the superset.
- The subset does not need to contain all elements of the superset.
- The subset may be defined with explicit or calculated elements as discussed below.

3.1.4 Element definition

Elements within sets can be entered explicitly or in the case of subsets may be defined with computations. Each case is covered below.

Explicit element definition
Element definition by computation

3.1.4.1 Explicit element definition

Explicit element definition involves the types of statements as above composed of

```
set
name of the set
optional explanatory text
/
element names followed by optional explanatory text.
between element names either a comma or a carriage return
/.
```

In element definition, one may cause multiple elements to be defined in sequence by using notation such as r1*r4 which causes definition of r1,r2,r3,r4. One can also place the number in other positions using for example 1u*10u which would define 1u, 2u, 3u etc. up to 10u. Additionally one may assign members to a set in decreasing order.

Examples:

(sets.gms)

```
SETs rj                              /x1,"x2 item",'x3*case'/
     ri                                    /r1 ,r2,r4*r10, 1a*200a/;
SET   mPROCESS          PROCESSES          /X1,X2,X3/;
SET   mcommodities            Commodities        /    corn  in bushels
                                                  Wheat in metric tons
                                                  milk  in pounds/ ;

SET years               items in decreasing order
```

© 2022 Prof. Bruce McCarl

```
                                                        /
bc2000*bc1,rr2009*rr0/;

      SET years2              decreasing  /"-20"*"-1"/;
```

3.1.4.1.1 Set definition through Tables

One can define set elements for sets that have 2 or more dimensions through tables. This is done using a set then a table command or a set table command where the table entries are numerical or yes/no.

Examples:

The following four definitions all have the same effect (**sets.gms**)

```
      set     Linkedbyroad2(origins,destinations)  Places linked by  roadways
                              /"NEW York" .Portland, "New York" .Houston,
                                boston.portland, boston.Houston/;

      set Table Linkedbyroad3(origins,destinations)  Places linked by  roadways
                        Portland London Houston
              "NEW York"   yes            yes
               boston      yes     No     yes;

      set Linkedbyroad4(origins,destinations)  Places linked by  roadways;
      Table   Linkedbyroad4(origins,destinations)  Places linked by  roadways
                        Portland London Houston
              "NEW York"   yes             yes
               boston      yes     No      yes;

      set Table Linkedbyroad5(origins,destinations)  Places linked by  roadways
                        Portland London   Houston
              "NEW York"   3209              1429
               boston      3180     0         1520;
```

Notes:

One can

- Define the set and elements in a tabular framework using the syntax Set Table.
- Define the set first using a set command without elements specified that later go back to fill in the elements with a table command.
- Define the set elements using the same convention as in the Data Entry chapter table command section (with entries aligned under column names) using either nonzero numeric entries to define active elements or yes/no.
- Three or more dimensional items are defined as in the Data Entry chapter, table command section.

© 2022 Prof. Bruce McCarl

3.1.4.2 Element definition by computation

Elements of subsets may also be computed. This is done by using a statement such as

```
Set("nameofelement")=yes;
```

which is equivalent to including a set element named nameofelement in the set declaration. One may also use computations to remove elements by using a statement like

```
Set("nameofelement")=no;
```

Examples:

(sets.gms)

```
Set   nSuperset                    /r1*r15/
      nSubset(nsuperset);
nSubset(nsuperset)=yes;
nSubset("r4")=no;

Set   nallitems                   /Corn,wheat,water,land/
      nCrop(nallitems)  ;
Parameter yield(nallitems)    /Corn 100,Wheat 40/;
nCrop(nallitems)$(yield(nallitems) gt 0)=yes;
```

Notes:

- By default all elements in a set without definition are undefined (set to no), so one only has to identify the items present (setting them to yes).
- Setting an element to no removes it from the set.
- Complex conditionals can be employed in defining the sets. Conditionals are discussed in the Conditionals chapter.
- Computed sets cannot be used to define the domain of data items. One must use the superset for the domain.
- Sometimes one will compute a set and set all elements to yes then go back and selectively remove items using a command setting them to no.
- More on set computations appears in the Calculating Items chapter.
- Calculated sets are called dynamic and cannot be used with the ord command. Leads and lags only work in conjunction with $offorder.

3.1.5 Multi dimensional sets

Sets do not need to be one dimensional and rather can be composed as sets of other sets where the basic notation to identify the presence of an element is

```
set  multidimset(set1name,set2name)
```

© 2022 Prof. Bruce McCarl

```
/set1elementname.set2elementname /
```

with the period separating the two set elements.

Examples:

(sets.gms)

```
Sets  Origins           Originating Places      /"New York", Boston/
      Destinations            Demand points     /"Portland",London, Houst
      Linkedbyroad(origins,destinations)  Places linked by  roadways
                         /"NEW York" .Portland, "New York" .Houston,
                           boston.Portland, boston.Houston/;
```

Notes:

- Up to 20 sets may be used to define a multidimensional set.
- These sets are useful in speeding up GAMS and making sure that unneeded cases like sending goods by truck across the ocean are not considered in the context of conditionals.
- A major use of multi dimensional sets is as a tuple in sums or conditions.
- Shorthand notation may be used to specify elements in multidimensional sets using parentheses. For example, the statement

  ```
  Set xx(origins,destinations) /boston.(houston,london)/;
  ```

  is the same as the statement

  ```
  Set xx(origins,destinations) /boston.houston,boston.london/;
  ```

- As of version 24.2 the alias may be used with multidimensional sets.
- One can add explanatory text to multidimensional sets as illustrated below where the text is in red

```
set   d1 /d1,d2/
              e1 /e1,e2/
              f1 /f1,f2/
              tuplewithexp(d1,e1,f1)/ d1.e1.f1 has text
                                    d2.e2.f2/
```

3.1.6 Domain checking

The GAMS compiler conducts 'domain checking,' with respect to subset definition and set element usage. Domain checking verifies that each element defined in a subset is in fact a member of the superset. It also insures that each referenced element of a set in GAMS calculations or other equations is in fact a member of the set associated with the definition of that location in parameters, variables, sets etc. When the items are not in the domain of the referenced set GAMS issues a compilation error and points to the missing element.

© 2022 Prof. Bruce McCarl

Examples:

(seterr.gms)

In the following case, the elements in **bold and blue** would pass the domain check, but the elements in **bold and red** would stimulate compiler errors because **Baltimore** is not an element of the set places and **seatle** is a misspelling.

```
Set places list of locations  /boston,Miami,seattle/;
Set place(places)       /boston,Miami,seatle,Baltimore/;
Parameter dataitem(places) / boston 5,Miami 8,seatle 4,Baltimore 3/;
```

Notes:

- Domain checking is automatic and is only suppressed under two circumstances
  - When the set in the position is either the universal set or is aliased to the universal set as discussed below.
  - When the $onwarning option is used to suppress domain checking.
- Domain checking finds misspellings and omitted elements and thus should be used as often as possible.
- Sets with calculated elements cannot be used in definition of item domains. This will generate a GAMS error. These sets are called dynamic sets.

3.1.7 Set element referencing

Set elements are referenced in calculations, equation specifications, loops and many other statements. GAMS statements ordinarily refer to either a single element or to every element in the set. Special provisions must be made to operate on more than one, but not all, elements of a set. Set referencing may also be controlled by tuples where multiple sets are referenced only for specially defined joint elements. Each of these cases is defined below.

Whole sets
Single elements
Operating over part of a set
Universal Set: * as a set identifier

3.1.7.1 Whole sets

GAMS ordinarily operates over every element in a set. Thus, the command (**sets.gms**)

```
Set II /i1*i4000/;
Parameter x(ii);
x(ii)=4;
```

will define every case of x associated with the set II to 4 and in this case 4000 of them.

© 2022 Prof. Bruce McCarl

Similarly the definition

```
X(ii)=y(ii) +3;
```

Will sequentially define every case of x in II to equal the associated case in y plus 3;

Also the following commands will each operate or define items for each case of II

```
Loop(II, z=z+y(Ii));
Z=sum(II,y(Ii));
Variable zz(Ii);
Equation eq(Ii);
```

3.1.7.2 Single elements

One can also specify an element name in quotes to cause GAMS to operate over just a single element of a set. Thus the command

```
x("i344")=4;
```

will only operate over the I344 element of x leaving the rest alone.

Similarly, the definition

```
X("i344")=y("i344") +3;
```

will only define the i344 case of x to equal the i344 case in y plus 3; The command

```
X(Ii)=y("i344") +3;
```

will set every case of x associated with the entries in Ii to the i344 case of y plus 3.

3.1.7.3 Operating over part of a set

Ordinarily one operates over each and every element of the reference set that an item is defined over. Thus in the case

```
NX(II,J)=4;
```

Each element of NX associated with every element of II in interaction with every element in J is operated over. However, there are cases where one wishes only to operate over part of those cases. In such a situation GAMS can be commanded to operate over part of the set through defined subsets, conditionals or tuples. Each case will be covered below.

3.1.7.3.1 Using subsets

There may be cases where one wishes to reference a priori known or calculated subsets. In such case,

© 2022 Prof. Bruce McCarl

one may define a subset, either explicitly or through calculation, then reference the item with respect to that subset. Consider the following example. (**sets.gms**)

```
Set    thisI            /i1*i10/;
Set    thisJ(thisI)     / i1,i3,i5/;
Parameter A(thisI)      /i7 5, i2 9, i3 11/;
Set wherea(thisI);
Wherea(thisi)$a(thisi)=yes;
Parameter nzz(thisi);
nZz(thisi)=5;
nZz(wherea)=-1;
nZz(thisj)=12;
```

In this case, the blue colored statements operate over subsets of thisI with the where a reference being over a calculated subset and the this j reference being over an explicitly defined subset.

3.1.7.3.2 Using conditionals

One may also operate over part of a set depending upon a conditional. For example the following statement

```
Z=sum(thisI$nzz(thisI),1);
```

would add up the number of elements in thisI that are associated with a nonzero value of nzz(thisI) as controlled by the conditional **$nzz(thisI)**. The chapter on conditionals covers a lot more cases and provides a fuller description.

Some particular forms of conditionals merit special mention in this document on sets. Conditionals can involve functions that return particular values depending on the position of elements in sets, the length of sets or the comparison of set elements to each other or text strings. These functions are defined in the section on functions below. Discussion of their use follows.

3.1.7.3.2.1 Sameas and Diag

Sameas and **diag** are functions that allow comparison of set names. Suppose I wish to add up shipments within cities. Further suppose I have an array move(origins,cities) giving the amount from origins to cities. In addition the within city are those in move(origins, cities) where the origin name is the same as the cities name. (**sets.gms**)

```
Alias(origins,cities);
Z=sum((origins,cities)$sameas(origins,cities),move(origins,cities));
```

or

```
Z=sum((origins,cities)$diag(origins,cities),move(origins,cities));
```

Equivalently a statement like

```
Z=sum((origins,cities)$(not sameas(origins,cities)),move(origins,cities));
```

© 2022 Prof. Bruce McCarl

adds up the shipments between cities.

One could also operate over particular elements using these commands

```
Available(resource)$(sameas(resource,"cropland") or
                     sameas(resource,"pasture"))..
  Sum(activity,usage(resource,activity)*xvar(activity))=l=
                      endowment(resource);
```

where the sameas command would compare the text for the element name for each elements of the set **resource** with the string **cropland** or **pasture** and if so operate over that part of the **resource** set.

3.1.7.3.2.2 Ord and Card

ORD and CARD are functions defined in the Conditionals chapter that allow knowledge of and special processing for the relative position of a set element within a set. Namely, I may also wish to do particular things if I am on the first or last or other elements of a set. The statement (**sets.gms**)

```
stock(t)$(ord(t) eq 1) = initial;
```

defines a constraint only for the first element of t.

The command

```
Carryout.lo(t)$(ord(t) eq card(t)) = final;
```

only operates for the last element of t.

3.1.7.3.3 Using tuples

A **tuple** refers to a set defined over other sets. The set may either be a one dimensional subset or a multidimensional set. Tuples are useful in calculations and in imposing conditionals.

Examples:

(sets.gms)

One can replace a sum that would go over all cases of a set with one that only operates over a subset. Namely in

```
mZ=sum(r(mi),mx(mi));
```

the index r(mi) only operates over those elements in mI that appear in the subset **r**.

Similarly in

```
mQ(i_am_a_tuple (mI,mj) =mx(mi)+my(mj);
```

© 2022 Prof. Bruce McCarl

the only the mi and mj cases which are operated over are those explicitly defined in the set named **i_am_a_tuple**.

Finally, note that when using a tuple on both sides of the equation that one does not need to explicitly enter the component sets as follows

```
mQ(i_am_a_tuple) =mQ(i_am_a_tuple )*1.5;
```

where mq is declared as mq(mi,mj) in **sets.gms** but so is i_am_a_tuple so the mi and mj can be left out of the replacement statement and the replacement will operate over all mi and mj cases in the tuple.

3.1.7.3.4 Defining a tuple with the matching and # operators

Mappings between tuples can be lengthy and inconvenient to enter via data statements plus difficult to compute. The matching operator (**:**) can be used to simplify definition and assignment. When using a matching operator one uses the general syntax

setsa:setsb

where elements of the **set or sets specified befoore the :** are matched with elements **of the set or sets specified after the colon** in the order both are specified in GAMS up until the matching is complete or all of the elements of one set or the other have been used**.** Namely the matching will follow the order of set elements in GAMS with the first element of one set matched with the first element of the second set etc.

For example, the statement (matchtuple.gms)

```
Set I / t1*t6:s3*s5 /
```

matches t1 with s3, t2 with s4 and t3 with s5. The elements t4, t5 and t6 are not matched because the elements in the second set specification are exhausted. The result is the same as the explicit set specification

```
Set j / t1.s3,t2.s4,t3.s5 /
```

One may also construct all combinations of the elements of 2 sets using notation involving the set name and a # as follows

```
sets h /h1*h5/, d /d1*d20/, dh(d,h) /#d.#h/;
```

and address whole sets in the matching operation again using the set name and # as follows

```
sets t /t1*t100/, tdh(t,d,h) /#t:#dh/, dht/#dh:#t/;
```

The resulting set tdh will then have the values:

```
t1.d1.h1, t2.d1.h2, t3.d1.h3 ..
```

while dht will have

© 2022 Prof. Bruce McCarl

```
d1.h1.t1,d1.h2.t2, d1.h3.t3 ...
```

An option statement also causes the matching to occur. Namely given the set definitions

```
set i1 /el1*el5/,j1/jel1*jel10/,k/ka,kb,kc/,l/l1*l200/
;
Set ijk(I1,j1,k), x(I1,j1,k,l);
```

Then using an option statement that contains the matching operator (:) also causes the matching to occur. Namely given the command

```
Option ijk(i1:j1,k), x(ijk:l);
```

Results in the set ijk being emptied then the set ijk being defined according to a matching of elements of I with j for each k In turn then the x set is defined with the elements of ijk matched with l.

3.1.8 Universal Set: * as a set identifier

Set references may be indefinite allowing any entries at all by referring to the universal set. This is done by either

- Using an * instead of a set name in an item definition, or
- Aliasing a set to the universal set (denoted by an *) and then using that set in item definitions.

In either case domain checking is suppressed and any entry whatsoever may be used without error.

Examples:

(sets.gms)

Here I use the universal set in a number of places

```
Set knownset /p1*p4/;
Alias (newuniverse,*);
Set a1(newuniverse);
Parameter  dataitem(*)  data without fixed set assignments /
                              Newitem1 1, newitem2 3/;
Parameter dd(newuniverse);
Dd(knownset)=4;
Dd("newone")=5;
Dataitem("newitem4")=dataitem("newitem1")*dataitem("newitem2");
A1("boston")=yes;
```

where the blue and bolded items are all associated with universal sets and no domain checking is going on and new elements can be freely introduced.

Notes:

- Use of universal sets for data input items is not recommended as spelling errors will not generally

© 2022 Prof. Bruce McCarl

be detected.

- GAMS will check in replacement statements to make sure specifically referenced elements have been defined and will give an error if not (setuniverr.gms). But this is not done in model equations.
- Sometimes this is useful in finding the sets over which data items are defined or in quickly formulating reports.
- The universal set is specified as ordered and ordered operators like lag, leads and ORD can be applied to any sets aliased with it.

3.1.9 Using set attributes

Set elements have attributes that may be recovered during execution. In particular there are 9 attributes that may be revered in a statement of a for

a(setname)=setname.attribute;

where

setname is the name of the set

attribute is one of the following

- **ord** which gives the position of the element in the current set so for the first element is at position one, the second at two etc.
- **rev** which gives the reverse position in the current set (card(i)-i.pos) so the last element is at reverse position zero, the last but one at one etc
- **pos** which gives the element position without the requirement that the set be ordered so for the first element is at position one, the second at two etc.
- **off** which gives the position in the current set less one (i.pos-1) so for the first element is at position zero, the second at one etc
- **uel** which gives the element position in the unique element list
- **val** which converts set element names that **happen to be numbers** into values.
- **len** which gives the length of the text for the set element name (a count of the number of characters)
- **first** which gives 1 for the first element of the current set and 0 for all others
- **last** which gives 1 for the last element of the current set and 0 for all others

Notes:

- **ord** is the same as the function **pos** and only works when the set is ordered otherwise it generates a compilation error.
- **pos** which gives the element position without the requirement that the set be ordered so for the first element is at position one, the second at two etc.

© 2022 Prof. Bruce McCarl

- **val** generates a number of the set element has a numeric counterpart (ie when the element text is "1" or "100" etc but not when any non numeric characters .excepting a decimal point are present in the set element name in which case an execution error occurs

Examples:

```
set id set to find attributes for / aa,'-inf',1,12,24,'13.14',inf /;
parameter report(id,*) gives set values;
         report(id,'value')    = id.val;
         report(id,'length')   = id.len;
         report(id,'offset')   = id.off;
         report(id,'position') = id.pos;
         report(id,'ord') = id.ord;
         report(id,'uel') = id.uel;;
display report;
```

which generates the output

```
**** Exec Error at line 188: Coud not extract number from element: aa
```

(this error is because element "aa" is non numeric)

```
----    194 PARAMETER report  gives set values

            value      length      offset    position         ord         uel

aa            UNDF       2.000                  1.000       1.000    4260.000
-inf          -INF       4.000       1.000      2.000       2.000    4261.000
1            1.000       1.000       2.000      3.000       3.000    4262.000
12          12.000       2.000       3.000      4.000       4.000    4263.000
24          24.000       2.000       4.000      5.000       5.000    4264.000
13.14       13.140       5.000       5.000      6.000       6.000    4265.000
inf           +INF       3.000       6.000      7.000       7.000    4266.000
```

Note aa has an undefined value

3.1.10 Finding sets from data

Sometimes it is desirable to find the set that characterizes an item then use it from then on. One may accomplish this by using an alias with the universal set and then compute set elements based on data using conditionals. This also can be accomplished through the load command drawing data from a GDX file.

Examples:

Consider the example **trnsprt.gms** where this is done using several steps. First I define sets without specifying elements (sources and places here) as equivalent to universal (unspecified) set.

© 2022 Prof. Bruce McCarl

```
alias(sources,places,*)
```

Then I enter data which contains an indicator of which set elements are valid entries in the set to be computed where in this case to be associated with the set sources the named place must have totalsupply.

```
table trandata (sources,places) data from spreadsheet
                    newyork     chicago    totalsupply
      seattle         2.5          1.7         350
      Sandiego        2.5          1.8         300
      totalneed       325           75
```

Now in preparation of set calculation I define subsets for the sets I will compute. These sets will be set to yes based on the data.

```
set source(sources)    sources in spreadsheet data
    destinaton(places) destinations in spreadsheet data;
```

Then I compute the set elements based on the data. In this case a source is defined (set to yes) if that location has an entry for totalsupply and a destination is defined if that place has an entry for totalneed.

```
source(sources)$(trandata(sources,"totalsupply"))=yes;
destinaton(places)$(trandata("totalneed", places ))=yes;
```

These sets can be used from then on.

Such computations are useful if a report has been specified with indefinite elements but needs to be manipulated or if one gets in a data table from elsewhere which defines the problem dimensions (set elements).

3.1.11 Using another name or an alias

There are occasions when one may wish to address a single set more than once in a statement. In GAMS this is done by giving the set another name through the ALIAS command as follows

```
ALIAS(knownset,newset1,newset2,...);
```

where each of the new sets will refer to the same elements as in the existing known set. As an alternative to this one can use the .Local notation but generally only in macros and only ith care..

Examples:

(sets.gms)

Suppose I have a two-dimensional data item that addresses the same set in both dimensions and I wish to compute the cost from each place to each other as a function of distance. To do this I use an alias as follows

```
Set place   /p1,p2/;
Alias(place,otherplace);
Table distplace(place,place)  distaces
          P1   p2
P1        0    4
```

© 2022 Prof. Bruce McCarl

```
P2         4    0;
Parameter cost(place,place) cost data;
Cost(place,otherplace)=1+5*distplace(place,otherplace);
```

Notes:

- As of version 24.2 the alias may be used with multidimensional sets.

3.1.12 Element order and capitalization in output

Set element ordering and capitalization are dictated by the general rules in GAMS for such items which is called the Unique Element List or UEL that is discussed in the Rules for Item Capitalization and Ordering chapter. The short answer from these rules is that the capitalization used is the first one seen in the program and the order is the order in which the names for the set elements first appear in the program.

3.1.13 Functions specifically referencing sets

There are four types of functions that are usable within GAMS that involve sets. These allow comparisons of set elements (sameas,diag), an indication of the relative position of a set element within a set (ord) and a count of the total number of elements within a set (card).

Ord
Card
Sameas
Diag

3.1.13.1 Ord

ORD(setelement) reports the position of his particular setelement within the overall set. Thus the command

```
thisX(ione)=ord(ione);
```

will set thisx(ione) equal to one for the first element in Ione, two for the second etc;

Notes:

- Ord only works with ordered sets.
- Ord refers to the relative position of each element in the set not necessarily the order in which they are typed. In particular the order may be different as determined by the rules for set ordering.

3.1.13.1.1 Ordered and Unordered sets

Ord and the leads and lags below only work on ordered sets unless the $command $offorder is specified. Such sets typically must have explicit element definitions and cannot contain calculated elements, thus being only sets with a priori specified values. Unordered sets are those that are not

© 2022 Prof. Bruce McCarl

ordered. The universal set is ordered and any set may be reported in ordered form using the special predefined tuple set SortedUels(*,*). For example, to write a set in sorted order:

```
alias(*,u);
loop(SortedUels(u,i),
    put / i.tl i.te(i) );
```

Sets with explicit elements are not always ordered if the elements are defined in two explicitly specified sets and are referenced out of order in the second one as discussed here.

Note sets with calculated elements are also sometimes called dynamic.

3.1.13.2 Card

CARD(setname) reports the count of the total number of elements within the set. Thus the command

```
number=card(i);
```

will set the parameter number equal to the count of the total elements in i.

Note card works with any sets whether they contain calculated elements or not.

3.1.13.3 Sameas

One may wish to do conditional processing dependent upon the text defining a name of a set element matching the text for a particular text string or matching up with the text for a name of a set element in another set. This can be done in GAMS using the sameas command. **SAMEAS (setelement,othersetelement)** or **sameas(asetelement,"text")** returns an indicator that is true if the text giving the name of **setelement** is the same as the text for **othersetelement** and a false otherwise. Similarly **sameas(asetelement,"texttotest")** returns an indicator that is true if the text giving the name of **asetelement** is the same as the **texttotest** and false otherwise. SAMEAS can also be used as a set.

Examples:

(sameas.gms)

The following **red** use of **sameas** will only permit the case of cityI and cityj to be part of the sum where the elements for both are **boston** and do not require the sets to be subsets of each other. The **blue** use will only operate for the element of I associated with the name "**new york**".

```
Set   cityI       / "new york", Chicago, boston/;
Set   cityj       /boston/;
Scalar ciz,cir,cirr;
ciZ=sum(sameas(cityI,cityj),1);
ciR=sum((cityI,cityj)$ sameas(cityI,cityj),1);
ciRR=sum(sameas(cityI,"new york"),1);
```

© 2022 Prof. Bruce McCarl

Note:

The above examples show that sameas can be used as a tuple or a multidimensional set.

3.1.13.4 Diag

DIAG(setelement,othersetelement) or **diag(asetelement,"text")** returns a number that is one if the text giving the name of **setelement** is the same as the text for **othersetelement** and a zero otherwise. Similarly **diag(asetelement,"texttotest")** returns a one if the text giving the name of **asetelement** is the same as the **texttotest** and zero otherwise.

Examples:

(diag.gms)

The following **red** use of **diag** will only be one for cityI and cityj where the elements for both are the same (**boston** in this case). The **blue** use will only be one for the element of cityI associated with the name "**new york**".

```
Set   cityI       / "new york", Chicago, boston/;
Set   cityj       /boston/;
Scalar ciz,cir,cirr;
ciZ=sum((cityi,cityj),diag(cityI,cityj));
ciRR=sum(cityi, diag(cityI,"new york"));
```

3.1.14 Indexing sets defined over time

Special features are included in GAMS for use with sets that represent time. These involve leads and lags in both equilibrium and non equilibrium settings and the use of special functions for particular time periods.

Leads and Lags: + / -
Circular or Equilibrium Leads and Lags: ++ / --
Element Position

3.1.14.1 Leads and Lags: + / -

Some problems involve sets defined over time covering years or quarters or months. Often when operating with such sets one may wish to define carryover relationships. For example, suppose beginning storage in a quarter equals ending storage in a previous quarter. These operators only work with ordered sets unless one uses the $ command **$On/OffOrder.**.

GAMS lead and lag features for set referencing are used as follows (**dynindex.gms**)

```
Stockbal(t).. endstock(t-1)=e=beginstock(t);
```

Where the **-1** notation references the previous time period to the current one, or equivalently

© 2022 Prof. Bruce McCarl

```
Stockbal2(t).. endstock(t)=e=beginstock(t+1);
```

Notes:

- The lead and lag can also use **+2** to go two periods into the future or +someothernumber for other leads and –somenumber for lags.
- When the case identified by t**-1** ot t**+1** etc does not exist the term is just skipped. Thus in the stockbal example above no lag t**-1** tem is defined for the first case of t and no t**+1** case is defined in stockbal2 equation for the last set element in t.

3.1.14.2 Circular or Equilibrium Leads and Lags: ++ / --

Some problems involve sets defined over time covering quarters or months. Often when operating with such sets one may wish to define carryover relationships where the year wraps around in an equilibrium fashion and beginning storage in January equals ending storage in December. These operators only work with ordered sets.

Examples:

The GAMS equilibrium lead and lag features for set referencing are used as follows (**dynindex.gms**)

```
Stockbal3(t).. endstock(t--1)=e=beginstock(t);
```

Where the **--1** notation references the previous time period to the current one and wraps to the last element when on the first element, or equivalently

```
Stockbal4(t).. endstock(t)=e=beginstock(t++1);
```

Notes:

- The lead and lag can also use ++n (--n) to go n periods into the future (past).
- When a case t**--1** or t**++1** etc does not exist the reference wraps restarting at the top or the bottom.
- The leads an lags ordinarily can only be used on sets with static contents (predefined) that are ordered. Tis can be relaced through use of $offorder.

3.1.14.3 Element Position

Dynamic models often lead one to need to specify initial, terminal and normal operating rules. For example, given a model defined over years one could want beginning storage in year one to equal Initial storage, ending storage in the last period to equal a fixed amount and initial storage in the years in between to equal carry out storage from the year before. This is commonly imposed using CARD and ORD. In such a case one could impose the following (**dynindex.gms**)

```
Storecarry(t)..  Beginstorage(t) =e= initial$(ord(t) eq 1) +endstorage(t-1);
Termstore(t)$(ord(t)=card(t))..    Endstorage(t)=e=finalstore;
```

© 2022 Prof. Bruce McCarl

3.1.15 Set Arithmetic

Arithmetic like set operations can be performed over sets that are a subset of a common superset to form set unions, intersections, complements, and differences

Unions
Intersections
Complements
Differences

3.1.15.1 Unions

Set unions can be formed using an addition type operation namely (**setarith.gms**)

```
Subset3(superset) = Subset1(superset) + Subset2(superset);
```

The membership of subset3 contains all elements that are either members of subset1 and or subset2. This operation is equivalent to the statements

```
Subset3(superset)=no; subset3(subset1)=yes; subset3(subset2)=yes;
```

3.1.15.2 Intersections

Set intersections can be formed using a multiplication type operation namely (**setarith.gms**)

```
Subset3(superset) = Subset1(superset) * Subset2(superset);
```

The membership of subset3 contains all elements that are members of both subset1 and subset2. This operation is equivalent to the statements

```
Subset3(superset)=yes$(subset1(superset) and subset2(superset));
```

3.1.15.3 Complements

Set complements can be formed using the not operator (**setarith.gms**)

```
Subset3(superset) = not Subset1(superset);
```

The membership of subset3 contains all elements that are not members of subset1. This operation is equivalent to the statements

```
Subset3(superset)=yes; subset3(subset1)=no;
```

3.1.15.4 Differences

Set differences can be formed using a subtraction type operation namely (**setarith.gms**)

```
Subset3(superset) = Subset1(superset) - Subset2(superset);
```

© 2022 Prof. Bruce McCarl

The membership of subset3 contains all elements that are in subset1 but not in subset2. This operation is equivalent to the statements

```
subset3(subset1)=yes; subset3(subset2)=no;
```

3.2 Data Entry

GAMS provides for four forms of data entry. These involve:

Scalars
Parameters
Table
Calculated data

3.2.1 Scalars

A SCALAR declaration is used to enter items that are not defined with respect to sets. The general form of the SCALAR entry is

```
scalar
  item1name             optional explanatory text    /numerical value/
  item2name             optional explanatory text    /numerical value/
     ...                                 ;
```

or

```
scalars
  item1name             optional explanatory text    /numerical value/
  item2name             optional explanatory text    /numerical value/
     ...                                 ;
```

Examples:

(scalar.gms)

```
scalar    dataitem      /100/;
scalar    landonfarm    total arable acres /100/;
scalars   landonfarms   /100/
          cost          /-10.02/
          pricecorn      1992 corn price per bushel  /2.20/;

scalars   a1 ,   a2 ,  a3 /5/;
scalar    withnodata  enter a scalar without data;
```

Notes:

- Scalar names plus the explanatory text must obey the rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.

© 2022 Prof. Bruce McCarl

- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- SCALAR or SCALARS can be used interchangeably.
- More than one named scalar is definable under a single scalar statement with a semicolon terminating the total statement.
- Multiple named scalars can be defined in a line set off with commas.
- Data do not have to be entered in the scalar statement, but rather can be defined later with replacement (=) statement calculations or assignments.
- Scalars are a specific input entry format for the general GAMS parameter class of items that also encompasses Tables.

3.2.2 Parameters

Parameter format is used to enter items defined with respect to sets. Parameter format is most commonly used with data items that are dependent on only one set (a vector) although multi set cases can be entered.

The general format for parameter entry is:

```
Parameter
  itemname(setdependency)  optional explanatory text
           /first set element name   associated value,

            second set element name  associated value,
                              ...   /;
```

or

```
Parameters
  itemname(setdependency)  optional explanatory text
           /first set element name   associated value,

            second set element name  associated value,
                              ...   /;
```

Examples:

(parameter.gms)

```
PARAMETER      c(j)      / x1     3    ,x2   2 ,x3    0.5/
               b(i)     / r1 10
               r2 3/;
PARAMETER
                  PRICE(PROCESS)     PRODUCT PRICES BY PROCESS
                        /X1 3,X2 2,X3 0.5/;
               RESORAVAIL(RESOURCE)  RESOURCE AVAIL
                        /CONSTRAIN1 10 ,CONSTRAIN2 3/;
Parameter      multd(i,j,k) three dimension /
                   i1.j1.k1 10 ,
```

© 2022 Prof. Bruce McCarl

```
                    i2.j1.k2 90 /;
parameters      cc(j), cb(i) /i1 2/;
parameter     hh(j) define all elements to 10 /set.j 10/;
```

Notes:

- Item names, the contained set element names plus the explanatory text must obey the rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Explanatory text and long explanatory parameter names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- PARAMETER or PARAMETERS can be used interchangeably.
- More than one named item is definable under a single parameter statement with a semicolon terminating the total statement.
- Multiple named items can be defined in a line of a parameter statement set off with commas.
- Data do not have to be entered in the parameter statement, but rather can be defined later with replacement (=) statement calculations or assignments.
- Items can be defined over up to 20 sets and thus one named item may be associated with numerous individual numerical values for elements of the parameter, each associated with a specific simultaneous collection of set elements for each of the named sets.
- When multi set dependent named items are entered then the notation is

  set1elementname▪set2elementname▪set3elementname etc with periods(▪) setting off the element names in the associated sets.
- All elements that are not given explicit values are implicitly assigned with a value of zero.
- Multiple entries can occur within one command using notation such as

  ```
  Parameter a(i) /(i1,i5) 3,(i6*i11) 5/;
  ```
- The referenced set elements must appear in the set the named item is defined over.
- Data for an element can only be defined once in a parameter statement.
- Parameters are an all-encompassing data class in GAMS into which data are kept including data entered as Scalars and Table. Parameters may also contain acronyms.
- One can specify values for all elements in a set using the set reference set.setname as in definition of hh above.

3.2.3 Table

TABLE format is used to enter items that are dependent on two or more sets.

The general format is

```
Table itemname(setone, settwo ... ) optional explanatory text
                  set_2_element_1   set_2_element_2
set_1_element_1      value_11          value_12
set_1_element_2      value_21          value_22;
```

© 2022 Prof. Bruce McCarl

More than two set dimensions can be entered as shown below.

Examples:

(tables.gms)

```
TABLE a(i,j) crop data
            corn  wheat cotton
    land      1     1      1
    labor     6     4      8      ;

Table RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
             Makechair Maketable Makelamp
 plantcap        3         2        1.1
 salecontrct     1                  1    ;

Table fivedim(i,j,k,l,m) fivedimensional
                  l1.m1  l2.m2
        i1.j1.k2   11    13
        i2.j1.k11   6    -3
+                 l3.m1   l2.m7
        i1.j1.k2    1      3
        i10.j1.k4   7      9;

Table avariant1(i,j,state) crop data
            cn.al wt.al cn.al cr.in wt.in cn.in
    land     1      1    1     1     1     1
    labor    6      4    8     5     7     2;

Table avariant2(i,j,state) crop data
                   al  in
    land.corn      1    1
    labor.corn     6    5
    land.wheat     1    1
    labor.wheat    4    7
    land.cotton    1    1
    labor.cotton   8    2;
```

Notes:

- Item names, explanatory text and the contained set element names must obey the item naming rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- Table statements must contain at least one data element. Ordinarily, if data are to be defined by subsequent replacement (=) statement calculations or assignments it is usually better to define that item with a parameter statement.

© 2022 Prof. Bruce McCarl

- Items in tables must be defined with respect to at least 2 sets and can be defined over up to 20 sets. Thus one item name may be associated with numerous individual numerical values for elements of the parameter, each associated with a specific simultaneous collection of set elements for each of the named sets.
- Tables are a specific input entry format for the general GAMS parameter class of items that also encompasses scalars.
- When more than two dimensional items are entered in Tables then the notation is

  set1elementname▪set2elementname▪set3elementname etc with periods(▪) setting off the element names in the associated sets. Multiple orders can be used as illustrated in avariant1 and avariant2 above.
- Alignment is important. Each numerical entry must occur somewhere below one and only one column name in the Table.
- All elements that are not given explicit values or have blanks under them are implicitly assigned to equal zero.
- Multiple entries can occur within one table command using notation such as

```
Table ta(i,j)
             J1      j2*j4
   (i1,i5)   3         1
  (i6*i11)   5         8;
```

- The referenced set elements must appear in the set the named parameter is defined over.
- Data for an element can only defined once in a table statement
- Tables that become too wide can be split and continued with a + (plus).
- Tables can also be used to define elements of sets as discussed in the sets chapter.

3.2.4 Calculated data

Data may also be entered through replacement or assignment statements. Such statements involve the use of a statement like

```
parametername(setdependency) = expression;
```

where the parameters on the left hand side must have been previously defined in a set, parameter or table statement.

Examples:

(caldata.gms)

```
scalar  a1;
scalars a2 /11/;
parameter   cc(j) , bc(j) /j2 22/;
a1=10;
a2=5;
```

© 2022 Prof. Bruce McCarl

```
cc(j)=bc(j)+10;
cc("j1")=1;
```

Notes:

- When a statement like cc(j)=bc(j)+10; is executed this is done for all elements in k so if j had 100,000 elements this would define values for each and every one.
- These assignments can be the sole entry of a data item or may redefine items.
- If an item is redefined then it has the new value from then on and does not retain the original data.
- Items in parameter or scalar coefficients without a data definition must have a calculated value specified before they can be used.
- Calculations do not have to cover all set element cases of the parameters involved (through partial set references as discussed in the Sets chapter). Set elements that are not computed over retain their original values if defined or a zero if never defined by entry or previous calculation.

3.3 Variables, Equations, Models and Solves

In GAMS one specifies and solves models using an identification of variables, and equations along with an equation specification, a model specification and a solve statement. Here I cover specification of model types excepting those involving MPSGE. Users interested in that model type should review the document **MPSGE** .

Variables
Equation
Model
Solve: Maximizing, Minimizing, and Using

3.3.1 Variables

A variable in GAMS identifies a quantity that can be manipulated in the solution of an optimization model or a system of simultaneous equations. Variables and their set dependency must be declared before they can be used.

Declaration
Variable attributes

3.3.1.1 Variable Declaration

The general syntax for variable declaration is

```
Variabletype
   firstvariablename(setdependency)  optional explanatory text
            /optional values for attributes/
   secondvarname(setdependency)      optional explanatory text
            /optional values for attributes/
```

© 2022 Prof. Bruce McCarl

...
;

where variabletype gives restrictions on the eligible numerical values for a variable. The following types are allowed

Variable type	Nature of restrictions on numerical values of variable
Variable	No restriction, variable value can range from minus to plus infinity.
Free Variable	Same as case just above.
Positive Variable	Non-negative values only, variable value can range from zero to plus infinity Same as nonnegative below.
Nonnegative variable	Non-negative values only, variable value can range from zero to plus infinity Same as positive above
Negative Variable	Non-positive values only, variable value can range from minus infinity to zero.
Binary Variable	Restricted to equal either zero or one in an integer programming setting.
Integer Variable	Integer values only; by default ranging from 0 to 100.
SOS1 Variable	A group of variables only one of which can be non zero and by default are positive. Details appear in the MIP chapter.
SOS2 Variable	A group of variables only two adjacent ones of which can be non zero and by default are positive. Details appear in the MIP chapter.
Semicont Variable	Semi-continuous, must be zero or above a given minimum level. Details appear in the MIP chapter.
Semiint Variable	Semi-integer, must be zero or above a given minimum level and integer. Details appear in the MIP chapter.

Each variable must fit into one of these cases. One may also declare all variables in the Variable or Free classes then redefine into one of the others. This is not recommended as it entails multiple declarations of the same item.

Examples:

(model.gms)

© 2022 Prof. Bruce McCarl

```
Variables
  Tcost                  'Total Cost Of Shipping- All Routes';
Binary Variables
  Build(Warehouse)              Warehouse Construction Variables;
Positive Variables
  Shipsw(Supplyl,Warehouse)     Shipment to warehouse
     tu/S1.a.l 5//
  Shipwm(Warehouse,Market)      Shipment from Warehouse
Nonnegative Variables
  Shipsm(Supplyl,Market)        Direct ship to Demand;
Semicont Variables
  X,y,z;
```

Notes:

- Variable names, the contained set element names plus the explanatory text must obey the rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- Any of the variable type commands can end with the word variable or variables.
- More than one named variable is definable in a single variable statement separated by commas or line feeds with a semicolon terminating the total statement.
- Multiple named variables can be defined in a single line of a variable statement set off with commas.
- No data may be associated with a named variable in a variable statement.
- Named variables can be defined over from 0 up to 20 sets and thus one variable name may be associated with a single case or numerous individual variables, each associated with a specific simultaneous collection of set elements for each of the named sets.
- Every optimization model must contain at least one unrestricted named variable (i.e. one defined with just the Variable or the Free Variable type).
- Such definitions indicate that these variables are potentially defined for every element of the defining sets (also called the domain). However the actual definition of variables does not occur until the .. equation specifications are evaluated as discussed later.
- The optional attribute data are discussed in the variable and equation attribute section.

3.3.1.2 Variable attributes

Variables have attributes that can be used in specifying bounds, starting values, scaling and integer programming priorities. The attributes also contain the solution level and marginal for the variable after execution of a solve statement. They are more extensively discussed in the Calculations chapter and represent:

Variable attribute	Symbol	Description
Lower bound	.lo	Lower bound for the variable. Set by the user either explicitly or through default values associated with the variable type.

© 2022 Prof. Bruce McCarl

Upper bound	.up	Upper bound for the variable. Set by the user either explicitly or through default values associated with the variable type.
Fixed value	.fx	A fixed value for the variable which if set results in the variable up and lo bounds being set to the value of the fx attribute.
Range	.range	The difference between the lower and upper bounds for a variable cannot be assigned but can be used in computations.
Activity level	.l	Solution level for the variable, also the current value or starting point. This attribute is reset to a new value when a model containing the variable is solved.
Marginal	.m	Reduced cost, simplex criterion or dual value marginal value for the variable. This attribute is reset to a new value when a model containing the variable is solved.
Scale factor	.scale	Numerical scaling factor for all coefficients associated with the variable providing the model attribute scaleopt is set to 1. This is discussed in the Scaling chapter.
Branching priority	.prior	Branching priority value used in integer programming models providing the model attribute prioropt is set to 1 as discussed in the MIP chapter. Also permits one to relax integer restrictions by setting .prior = +inf regardless of prioropt setting.
Slack upper bound	.slackup	Slack from variable upper bound. This is computed as x.slackup = max (x.up-x.l,0)
Slack lower bound	.slacklo	Slack from variable lower bound. This is computed as x.slacklo = max(x.l-x.lo,0)
Slack	.slack	Smaller slack from variable bounds. This is computed as x.slack = min (x.slacklo,x.slackup)
Infeasibility	.infeas	Amount that variable is infeasible falling below its lower bound or above its upper bound. This is computed as x.infeas = - min(x.l-x.lo,x.lu-x.l,0)

The user distinguishes between these attributes by appending a suffix to the variable name. Values may be assigned as defined here.

Examples:

(model.gms)

```
 Shipsw.up(Supplyl,Warehouse)=1000;
 Shipwm.scale(Warehouse,Market)=50;
 Shipsm.lo(Supplyl,Market)$(ord(supplyl) eq 1 and
                                    ord(market) eq 1)=1;
totalship=
  sum((supplyl,market)   ,shipsm.l(supplyl,market))
+sum((supplyl,warehouse),shipsw.l(supplyl,warehouse))
+sum((warehouse,market) ,shipwm.l(warehouse,market));
```

© 2022 Prof. Bruce McCarl

Notes:

- When variables are used in display statements you must specify which of the attributes should be displayed. Appending the appropriate suffix to the variable name does this i.e.

```
display x.m,y.l,z.scale;
```

  No set element dependency specification appears.

- Wherever a variable appears in a GAMS calculation statement, the attribute desired must be specified.
- The only place where a variable name can appear without a attribute suffix is in a variable declaration, or a .. equation specification, which is discussed below.
- One can use the .L and .M attributes of variables to construct reports on problem solution as discussed in the Improving Output via Report Writing chapter.
- One can assign values to these items in data statements as discussed here.

3.3.1.2.1 Assigning variable and equation attributes

One may specify initial values for the attributes of equations and variables. Those new data statements follow the syntax for parameters or tables by adding an additional dimension to specify the specific data attribute but appear in the context of the variable and equation definitions or a Table statement.

The items that can be entered are the upper, lower and fixed bounds (.up, .lo, .fx); starting values (.l); scaling factors (.scale); marginals (.m); and priorities (.prior for variables only).

The format is an extension of the variable and equation commands of one of two forms.

- The conventional variable and equation statements can be augmented with a parameter like section where values are enclosed in between /'s as follows

```
 Variable x1(j) my first  / j1.up 10 , j1.lo 5, j1.scale 20, jl.l 7, j1.m 0 /
;

 Equation landconstrain(landtype) my land constraints
                      / cropland.scale 20, cropland.l 7, cropland.m 100 /;
```

- A table structure can be used The conventional variable and equation statements can be augmented with a parameter like section where values are enclosed in between /'s as follows

```
variable table x(i,j) initial values
                             l      m
     seattle.  new-york    50
     seattle.  Chicago    300
     san-diego.new-york   275
     san-diego.chicago         0.009;
```

© 2022 Prof. Bruce McCarl

```
Equation table landconstrain(landtype) my land constraints
              Scale   l   m

    cropland    20     7  100
    Pasture     10     6   30;
```

3.3.2 Equation

An equation in GAMS identifies a relationship in the model to be optimized or solved which is one of the constraints that must be satisfied in choosing the solution levels for the variables. Equations and their set dependency must be declared before their exact form can be specified.

Declaration
.. Equation specifications
Equation attributes

3.3.2.1 Equation Declaration

The general syntax for equation declaration is

```
Equation     firstequationname(setdependency)  optional explanatory
text
               /optional values for attributes/
           secondeqname(setdependency)      optional explanatory text
               /optional values for attributes/

                 ...
 ;
```

or

```
Equation     firstequationname(setdependency)  optional explanatory
text
               /optional values for attributes/
           secondeqname(setdependency)      optional explanatory text
               /optional values for attributes/

              ...
 ;
```

Example:

(model.gms)

```
Equations
        tcosteq                     total cost accounting equation
        supplyeq(supplyl)           limit on supply available at a
supply point
        demandeq(market)        minimum requirement at a demand market
        balance(warehouse)      warehouse supply demand balance
        capacity(warehouse)           warehouse capacity
```

© 2022 Prof. Bruce McCarl

```
                    /a.scale 50,a.l 10,b.m 20/
configure                       only one warehouse;
```

Notes:

- Item names, the contained set element names plus the explanatory text must obey the rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- Equation or equations may be used interchangeably.
- More than one named item is definable under a single equation statement separated by commas or line feeds with a semicolon terminating the total statement.
- Multiple named items can be defined in a single line of an equation statement set off with commas.
- Data may not be associated with a named equation in an equation statement.
- Equations can be defined over from 0 up to 20 sets. Thus one item name may be associated with a single case or numerous individual equations in the model, each associated with a specific simultaneous collection of set elements for the named sets.
- There are no modifying keywords preceding Equations as there are with variables.
- Such definitions indicate that these equations are potentially defined for every element of the defining sets (also called the domain). However the actual definition of variables does not occur until the .. equation specifications are evaluated as discussed later.

3.3.2.2 .. Equation specifications

While the equation section names the equations it contains nothing about their algebraic structure. That structure is defined in the specification or .. relationships within GAMS. In particular, each equation defined must be matched by a .. command which contains it's exact algebraic structure. The syntax for an equation specification is

```
equation name(setdependency)$optional logical condition . .

    lhs_equation_terms   equation_type   rhs_equation_terms;
```

where

- equation name(setdependency) must have already been named in an equation declaration.
- The set dependency specification must either match the sets used in the declaration, or use sets which are subsets thereof or elements thereof.
- The $optional logical condition is optional and is discussed in the Conditionals chapter.
- The definition always contains two dots '..' .

Examples:

(model.gms)

© 2022 Prof. Bruce McCarl

```
tcosteq..
 tcost =e=
  sum(warehouse,dataw(warehouse,"cost")/dataw(warehouse,"life")
              *build(warehouse))
 +sum((supplyl,market)   ,shipsm(supplyl,market)
                          *costsm(supplyl,market))
 +sum((supplyl,warehouse),shipsw(supplyl,warehouse)
                          *costsw(supplyl,warehouse))
 +sum((warehouse,market) ,shipwm(warehouse,market)
                          *costwm(warehouse,market));

 supplyeq(supplyl)..    sum(market, shipsm(supplyl, market))
                      + sum(warehouse,shipsw(supplyl,warehouse))
                       =l= supply(supplyl);

 demandeq(market)$ demand(market)..
                 sum(supplyl, shipsm(supplyl, market))
                   +  sum(warehouse, shipwm(warehouse, market))
                      =g= demand(market);

 balance(warehouse)..      sum(market, shipwm(warehouse, market))
                                  - sum(supplyl,shipsw(supplyl,warehouse))
                           =l= 0;

 capacity(warehouse)..  sum(market, shipwm(warehouse, market))
                            -build(warehouse)*dataw(warehouse,"capacity")
                           =l= 0 ;

 configure..            sum(warehouse,build(warehouse)) =l= 1;
```

Here GAMS will operate over all the elements in the sets in each term. For example, in the tcosteq equation GAMS will add up the term shipsm(supplyl,market)*costsm(supplyl,market) for all pairs of the set elements in supplyl and market. Similarly the equation capacity(warehouse) will define a separate equation case for each element of warehouse and within the equation for each paricular case of warehouse only the elements of shipwm(warehouse, market) associated with that warehouse will be included in the term sum(market, shipwm(warehouse, market)).

Notes:

- The algebraic terms of the equation (lhs equation terms, rhs equation terms) can be constants or algebraic expressions at least one of which contains variables.
- Terms containing variables can appear on either or both of the right or left hand side of the algebraic part of the equation specification.
- The equation type specifies the type of equation restriction with the following types allowed:

Symbol Used	Equation Type	Nature of restriction
=e=	Equality	**eq.term 1 = eq.term 2**
=g=	Greater than or	**eq.term 1 ≥ eq.term 2**

© 2022 Prof. Bruce McCarl

	equal to	
=l=	Less than or equal to	**eq.term 1 ≤ eq.term 2**
=n=	No specification	Rarely used but can occur in MCP models.
=x=	External defined	Equation defined by external program as discussed in Links to Other Programs Including Spreadsheets chapter.
=c=	Conic	See the Mosek solver manual or the GAMS notes on conic models a *http://www.gams.com/conic/*.
=b=	Boolean	A logic equation as discussed below

- Equation specification .. statements can carry over as many lines of input as needed.
- Blanks can be inserted to improve readability.
- An equation, once defined, cannot be altered or re-defined. If one needs to change the logic, a new equation with a new name will have to be defined.
- Parameter data or calculated sets incorporated in the equations may be changed by using assignment statements (see the Calculating Items chapter).
- Equation specifications end with a ;
- The equations and variables in a model are defined by the evaluation of the .. equation specifications. If conditionals or subsets used so that the entire domain of the defining sets is not covered the variables and equations will not be present for all elements of the domain. This given a constraint resource(j) and a subset i(j) where there are elements in j that are not in i and a .. command for resource(i).. means that resource equations will only be defined for the cases of j that also exist in i, not all that are in j.
- The only variables that will be defined for a model are those that appear with nonzero coefficient somewhere in at least one of the equations defined by the .. equations.
- More examples are in the GAMS test library
- Logic equations use Boolean algebra and have to evaluate to TRUE (or 1) to be feasible. The Boolean functions available in GAMS and the default order of precedence of the operators are given in the **Table below**.

Function	Operator	Alternative Notation	Return Values	Order of Precedence
Negation	not x	bool_not(x)	returns 1 if x=0 else 0	1
Logical Conjunction	x and y	bool_and (x,y)	returns 1 if x=y=1 else 0	2
Logical disjunction	x or y	bool_or(x,y)	returns 0 if x=y=0 else 1	3
Exclusive disjunction	x xor y	bool_xor (x,y)	returns 1 if exactly one argument is 1, else 0	3

© 2022 Prof. Bruce McCarl

Material implication	x imp y or x -> y	bool_imp (x,y)	returns 0 if x=1 and y=0 else 1	3
Material equivalence	x eqv y or x <=> y	bool_eqv (x,y)	returns 0 if exactly one argument is 0, else 1	3

Note that in the table above x and y are binary variables.

There are three ways to declare and define logic equations:

1. The logic equation is declared using the keyword `Logic Equation` and the definition contains only Boolean algebra symbols.

2. The logic equation is declared like any other equation using the keyword `Equation` and in the definition the symbol =b= appears indicating that it is a logic equation.

3. This is a combination of the first two options: the equation is declared with the keyword `Logic Equation` and defined using the symbol =b=.

The following example demonstrates the first way to declare and define a logic equation. It is adapted from the food manufacturing

problem foodemp. In this problem the blending of oils is modeled.

```
Sets
   m             "planning period (month)"    / m1*m6 /
   p             "raw oils"                   / v1*v2, o1*o3 /
Variables
   induse(m,p)   "indicator for usage of raw oil per month" ;
Binary variable  induse;

Logic Equation
   deflogic(m)  "if some vegetable raw oil is used we also need to use the non-
   vegetable oil o3" ;

deflogic(m)..  induse(m,'v1') or  induse(m,'v2') -> induse(m,'o3');
```

The variable `induse` is a binary variable, it can only take the values 0 and 1. The equation ensures that in an optimal solution if either vegetable oil v1 or vegetable oil v2 is blended in a product, then non-vegetable oil o3 is also blended in that product.

An alternative formulation of the equation `deflogic` using the =b= notation is given below.

```
deflogic(m)..  induse(m,'v1') or  induse(m,'v2') -> induse(m,'o3') =b= 1;
```

Note that the value of 1 on the right-hand side means that the logic expression on the left-hand side must evaluate to `TRUE` in a feasible solution. To illustrate further, we could negate the left-hand side expression using the logic operator `not` and then the right-hand side would have to evaluate to zero or `FALSE` to yield the same result as above. The respective

© 2022 Prof. Bruce McCarl

equation definition follows.

```
deflogic(m)..  not (induse(m,'v1') or  induse(m,'v2') -> induse(m,'o3')) =b= 0;
```

Note that currently logic equations are allowed only in models of the GAMS model type EMP.

3.3.2.3 Equation attributes

Equations have attributes that can be used in specifying starting values, and scaling. The attributes also contain the solution level and marginal for the equation after execution of a solve statement. They are extensively discussed in the Calculations chapter and represent:

Equation attribute	Symbol	Description
Lower bound	.lo	Negative infinity or the right hand side of a =g= or =e= equation.
Upper bound	.up	Positive infinity or the right hand side of a =l= or =e= equation.
Equation level	.l	Optimal level for the equation which is equal to the level of all terms involving variables.
Marginal	.m	Dual, shadow price or marginal value for the equation. This attribute is reset to a new value when a model containing the equation is solved.
Scale factor	.scale	Numerical scaling factor that scales all coefficients in the equation providing the model attribute scaleopt is set to 1. This is discussed in the Scaling GAMS models chapter.

The user distinguishes between these attributes by appending a suffix to the equation name.

Examples:

(model.gms)

```
SUPPLYEQ.scale(SUPPLYL)=33;
marg(supplyl)=SUPPLYEQ.m(SUPPLYL);
```

3.3.2.4 Assigning equation attributes

This topic is discussed here.

3.3.3 Model

Models are objects that GAMS solves. They are collections of the specified equations and contain variables along with the upper and lower bound attributes of the variables. The model statement identifies and labels them so that they can be solved. Three fundamental types of models can be identified.

1 Optimization models (LP, NLP, MIP, MINLP, ...) where the model statement identifies the

© 2022 Prof. Bruce McCarl

equations present in the model.

2 Simultaneous systems of equations (CNS) that are to be solved where the model statement identifies the equations present in the model.

3 Mixed complementary problems (MCP) where the model statement identifies the equations present in the model and their complementary relationships with the problem variables.

The basic form of the model statement is

```
Model Modelname  optional explanatory text / model contents /;
```

or

```
Models Modelname  optional explanatory text / model contents /;
```

Notes:

- Modelname, plus the explanatory text must obey the rules presented in the Rules for Item Names, Element names and Explanatory Text chapter.
- Labels and long explanatory names should be used where possible as argued in the Writing Models and Good Modeling Practices chapter.
- Model or Models can be used interchangeably.
- More than one named model is definable under a single Model statement separated by commas or line feeds with a semicolon terminating the total statement.
- Model contents can be defined in several different ways.
  - The notation /all/ includes all equations seen before the model statement within the named model. (**model.gms**)

```
Model warehousel Warehouse location model /all/;
```

  - In optimization and nonlinear system models one can list the equations to be included as follows (**model.gms**)

```
Model warehousel2   Warehouse location model
    /tcosteq,supplyeq,demandeq,balance,capacity,configure/;
```

  - One may include the names of models that were previously defined. Thus for example:

```
Model     one       first model                    /
TCOSTEQ,SUPPLYEQ,DEMANDEQ /
          two       second model that nests first  / one, balance /
          three     third model that nests first and second / two,
capacity,configure /;
```

In turn model two will now contain equations TCOSTEQ,SUPPLYEQ,DEMANDEQ while model three will contain all of model two plus equation balance. Similarly model three will have all of model two plus the two specified equations.

  - One may include + and - to augment or remove items relative to models that were previously defined. Thus for example:

© 2022 Prof. Bruce McCarl

```
Model four      fourth model that includes what is in model one and three / one
+three /
                five      fifth model includes equations in WAREHOUSEL2 but not
those in model one / WAREHOUSEL2-one /
                six       sixth model includes those in model WAREHOUSEL2 but
not equation configure/ WAREHOUSEL2-configure /
```

In turn model four will contain the union of all equations in models one and three while five will contain what is in the WAREHOUSEL2 model less the equations in model one. Finally model six will contain what is in WAREHOUSEL2 less the equation configure.

This shows both **model** and **equation** names can be used in association with the + and - symbols..

- In selected model types (MCP, MPEC, RMPEC) including mixed complementary models one can list the equations to be included and their **complementary variables** as follows (**mcp.gms**)

```
Model qp6 Michael Ferris example of  MCP
/ d_x?x,  d_w?w,  retcon?m_retcon,
     budget?m_budget, wdef?m_wdef  /;
```

where the notation has

equation name?complementary variable name

- Note this notation will allow GAMS to compile without errors in all model types but will not be an active part of the specification except in MCP, MPEC, and RMPEC models.

- In MCP models one can use the /all / notation to add all equations and match up complementary variables provided that all the equations are of the form =e= or =n= and the variables are all free with the problem being square with free variables defined with exactly matching subscripts for each equation (**kormcp.gms**). However this is somewhat risky as the model does not choose variables that are really related to equations and much of the problem structure is not conveyed to the solver. It is generally better to specify the complementary pairs in the model statement.
- In MCP problems one need only define part of the complementary relations and the remaining ones can be filled in by GAMS and the solvers providing a free variable can be found for each of the remaining equations and that the system is square. The equation names and possible complementary variable names included in the model contents specification need to be named variables and equations. Again, this is risky.
- Set membership is not included in the model contents field.
- The terms in an equation are recomputed every time a solve statement is executed as discussed in the Calculating Items chapter.
- The MCP problem structure imposes particular requirements on specification of the equations in the problem.
  - It is always acceptable to write the equations defining the problem with =N= relations. In

© 2022 Prof. Bruce McCarl

this case, the sign of the associated equation is implied by the equation/variable matching and the variable bounds.

- If a variable is complementary with an equation and that variable has lower bounds only (e.g if the variable in GAMS terms is a positive variable) then it is acceptable to write the complementary equation with =G= relations, since this is consistent with the constraint implied by the lower bound on the variable.
- A variable bounded only above can be matched with =L= equations.
- Free variables without bounds can be matched with =E= equations.

These restrictions are elaborated on in the Model Types and Solvers chapter.

3.3.3.1 Model attributes

A number of attributes of models may be accessed by the user in the form of numerical values. These include three fundamental types of items.

- Attributes that contain information about the results of a solver performed, Solve statement generated, solution of a model.
- Attributes that tell a solver or GAMS to use of certain features.
- Attributes that pass information to the solver or GAMS giving various setting that are also subject to option statement settings.

The general way these are used is as follows.

```
X=modelname.attribute;
Modelname.attribute=3;
```

where modelname is the name used in a model statement and attribute is one of the items listed below. More specifically, given the modelname is transport then statements like

```
x=transport.modelstat;
transport.holdfixed=1;
transport.bratio=1;
```

3.3.3.1.1 List of attributes

Here a brief list of the available attributes is presented. The attributes are extensively covered in the Model Attributes chapter. An * below indicates this attribute is also controlled by a GAMS option command.

Attribute	Main Usage relative to Solve		Notes
	Pre	Post	

© 2022 Prof. Bruce McCarl

Bratio*	X		Controls whether to discard advanced basis
Cheat	X		Minimum improvement in MIP objective value
Cutoff	X		Limit on acceptable MIP objective value
Domlim	X		Number of numerical errors allowed during solution
Domusd		X	Number of equation evaluation numerical errors encountered in solving
Fddelta	X		Step size for numerical derivatives
Fdopt	X		Controls methods of numerical Hessian calculation
Holdfixed	X		Reduces problem size by eliminating fixed variables
Iterlim*	X		Iteration limit which as of 23.1 has a default value of 2 billion
Iterusd		X	Number of iterations to solve problem
Limcol*	X		Limit on number of variables in output
Limrow*	X		Limit on number of equations in output
Marginals		X	Whether or not the solver provides a dual solution.
Mcprholdfx			An attribute if set to one results in a printout of the rows in an MCP problem that are complementary with variables that are held fixed by a .fx command when the holdfixed command is employed. More discussion on this appears here.
Modelstat		X	Model solution status. Values of 1 or 2 or 8 generally denotes optimal solution or with CNS values of 15 and 16., Other values denote infeasible, unbounded, solver failure etc. Consult solver manuals or the modelstat table for more.
Nodlim	X		Node limit in MIP
Numdepnd		X	Number of dependencies in a CNS model
Numdvar		X	Number of discrete variables in the model

© 2022 Prof. Bruce McCarl

Numequ		X	Number of total equations in the model
Numinfes		X	Number of infeasibilities in the solution
Numnopt		X	Number of non-optimalities in the solution
Numnz		X	Number of non-zero entries in the model coefficient matrix
Numunbd		X	Number of unbounded variables in the solution
Numvar		X	Number of single variables in the model
Objest		X	Bound on best possible objective value far a MIP as derived by the solver
Optca*	X		Max absolute MIP optimality gap
Optcr*	X		Max relative MIP optimality gap
Optfile	X		Causes use of solver options file
Prioropt	X		Causes use of MIP priorities
Reslim*	X		Max time available to solve in seconds
Resusd		X	Time in CPU seconds the solver used to solve the model
Robj		X	Relaxed objective value from MIPs when solve doesn't work
Scaleopt	X		Use user defined internal scaling factors
Solprint*	X		Controls solution print in LST file. A related set of solprint constants is also present.
Solveopt	X		Merge or replace solution information
Solvestat		X	Solver termination status. Value of 1 denotes normal termination, larger values denote iteration limits, solver failure etc. A full list appears here.
Sysout*	X		Expands solution output
Tmodstat		X	Can be used in put statements to give problem optimality

© 2022 Prof. Bruce McCarl

			status text
Tolinfrep	X		Sets the tolerance for marking infeasible in the equation listing. The default value is 1.0e-6
Tolproj	X		Sets tolerance for filtering marginals and variable/equation levels (i.e. setting marginals within a tolerance to 0 and projecting levels to their lower or upper bound) when they are within the tolerance when reading then outputting a solution, default is 1e-8. Levels reset are reported in solution report as projected. Marginals are only reset for basic items.
Tryint	X		Try to make current solution integer and use it in MIP solve
TryLinear	X		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 modelname.trylinear=1. Default value is zero. 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.
Tsolstat		X	Can be used in put statements to give problem optimality status text
Workspace*	X		Amount of CPU memory to allocate

3.3.4 Solve: Maximizing, Minimizing, and Using

Various types of problems can be solved with GAMS. The type of the model must be specified so GAMS can choose the appropriate solver to use as must some information about the model, and, if relevant, direction of optimization. This is done through a Solve statement. The general syntax for the Solve statement depends on model type and is

For optimization model types (LP, NLP, MIP, MINLP etc.) the Solve is one of the following 4 forms

```
Solve modelname maximizing var name using model type ;
Solve modelname minimizing var name using model type ;
Solve modelname using modeltype maximizing var name ;
Solve modelname using modeltype minimizing var name ;
```

with the possible addition of the words when GUSS is being used

```
SCENARIO tuplename ;
```

while for MCP and CNS model types which involve solution of equations systems we use

© 2022 Prof. Bruce McCarl

```
Solve modelname using modeltype ;
```

where

- **modelname** is the name of a model specified in a Model statement somewhere earlier in the program.
- **the key word maximizing** or **minimizing** is used to identify the direction of optimization.
- **var name** is the name of an unrestricted in sign variable (free or one declared as a variable only) which is declared and in a .. equation specification for at least one equation in the model and is the item to be maximized or minimized. Often such variables must be added as covered in the Tutorial chapter.
- **modeltype** which is one of the following types of problems.

GAMS **model type**	Model Type Description	Requirement
LP	Linear Program	Optimization problem which cannot contain nonlinear terms or discrete (binary or integer) variables
NLP	Non Linear Program	Optimization problem which contains smooth nonlinear terms, but not discrete (binary or integer) variables
QCP	Quadratically Constrained Program	Optimization problem which can contain quadratic nonlinear terms, but cannot contain other types of nonlinear terms or discrete (binary or integer) variables.
DNLP	Discontinuous Non Linear Program	Optimization problem which contains non smooth nonlinear terms with discontinuous derivatives, but not discrete (binary or integer) variables
MIP	Mixed Integer Program	Optimization problem which contains discrete (binary or integer) variables, but does not contain nonlinear terms
RMIP	Relaxed Mixed Integer Program	Optimization problem which contains binary, integer, SOS and/or semi variables, but does not contain nonlinear terms and has discrete/SOS/Semi variable requirement relaxed
MINLP	Mixed Integer Nonlinear Program	Optimization problem which contains smooth nonlinear terms and discrete (binary or integer) variables
RMINLP	Relaxed Mixed Integer Nonlinear Program	MINLP optimization problem with the binary and integer restrictions relaxed
MIQCP	Mixed Integer Quadratically Constrained Program	Optimization problem which can contain quadratic nonlinear terms, but cannot contain other types of nonlinear terms

© 2022 Prof. Bruce McCarl

RMIQCP	Relaxed Mixed Integer Quadratically Constrained Program	Optimization problem which can contain quadratic nonlinear terms and has discrete variable requirement relaxed.
MPEC	Mathematical Programs with Equilibrium Constraints	A difficult problem type for which solvers are just now under development and is the subject of a section on *gamsworld.org*.
RMPEC	Relaxed Mathematical Program with Equilibrium Constraints	A difficult problem type for which solvers are just now under development and is the subject of a section on *gamsworld.org*
MCP	Mixed Complementarity Problem	A problem solving a nonlinear system of equations which contains one to one complementary relationships between all of an equal number of variables and equations
CNS	Constrained Nonlinear System	A problem solving a square, possibly nonlinear system of equations, with an equal number of non-fixed variables and constraints
MPSGE	General Equilibrium	Not actually a model type but mentioned for completeness see **MPSGE**
EMP	Extended Mathematical Program	A family of mathematical programming extensions.

The exact form of these problem types is discussed in the Model Types and Solvers chapter.

- `SCENARIO` tells GAMS to use the GUSS scenario handler
- `tuplename` tells GUSS what to do when activated

Examples:

model.gms , mcp.gms , korcns.gms (from the GAMS model library) , resource.gms and GUSSRISK.gms

```
Solve warehousel2 using MIP minimizing tcost;
Solve qp6 using MCP;
Solve model1 using cns;
Solve resalloc using lp maximizing profit;
SOLVE EVPORTFOL USING NLP MAXIMIZING OBJ SCENARIO GUSSDICT ;
```

3.3.4.1 Actions on executing solve

GAMS does not directly solve problems rather other programs are used to solve the problem. However, GAMS does generate the problem in a form that is ready for the solver. In doing this several things happen:

- GAMS checks that the model is in fact the **type** the user thinks it is, and issues explanatory

© 2022 Prof. Bruce McCarl

error messages if it discovers a model beyond the solution capabilities of the solver to be used. For example the presence of nonlinear terms in a supposedly LP model.

- A solver is chosen which is either the
  - Default solver for that problem type
  - One specified on the GAMS command line
  - Solver chosen by an option statement.
- Optimization models are checked to see that the **objective variable is a scalar** (not defined over any sets) and of the variable type **free**, and appear in at least one of the equations in the model.
- MCP models are checked for appropriate complementarity and squareness.
- All equations in the model are checked to insure they have been defined.
- All sets and parameters used in the equations are checked to insure they have had values assigned.
- The model is translated into the representation required by the solver to be used.
- LIMROW and LIMCOL output is produced and written to the output file (Equation Listing, etc) as shown in the Standard Output chapter.
- GAMS verifies that there are no errors such as inconsistent bounds, inconsistent equations or unacceptable values (for example Na or Undf) in the problem. Any errors detected at this stage cause termination with an execution error reported as discussed in the execution errors chapter.
- GAMS passes control to the solution subsystem and waits while the problem is solved.
- GAMS collects back information on the solution process from the solver and loads solution values back into the memory. This causes new values to be assigned to the .l and .m variable and equation attributes for all individual equations and variables in the model plus the post solution model attributes. The procedure for loading back in the .l and .m data is controlled by the solveopt model attribute and option.
- A row by row and column by column listing of the solution is provided unless suppressed by the solprint model attribute or option.

3.3.4.2 Programs with multiple solve statements

Multiple solve statements can be present in a program.

Example:

(PROLOG.gms , Risk.gms)

```
solve nortonl using nlp maximizing z;
solve nortonn using nlp maximizing z;
solve nortone using nlp maximizing z;

loop (raps,rap=riskaver(raps);
            solve evportfol using nlp maximizing obj ;
```

© 2022 Prof. Bruce McCarl

```
var = sum(stock, sum(stocks,
    invest.l(stock)*covar(stock,stocks)*invest.l(stocks))) ;
output("rap",raps)=rap;
output(stocks,raps)=invest.l(stocks);
output("obj",raps)=obj.l;);
```

Notes:

- If you have to solve sequences of expensive or difficult models, you should consider using save and restart to interrupt and continue program execution.
- When more than one solve statement is present, GAMS uses information from the previous solution to provide a starting point for the next solution (see the Basis chapter for a discussion).
- 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 (**risk.gms**).

3.3.4.2.1 Multiple solve management - merge replace

When multiple solves are present one needs to pay attention to the way that GAMS manages solutions. In particular, when multiple models are solved GAMS by default merges subsequent solutions in with prior solution. This is not an issue if all the models operate over the same set of variables. However, if different variables appear in the solved models (due to recursive procedures, different equation inclusion or $ conditionals that can eliminate variables) then one should be aware that GAMS permits the user to modify the solution management procedure. This attribute tells GAMS how to manage the model solution when only part of the variables are in a particular problem being solved. In particular, the solution can either be merged with the prior solution for all variables or it can replace "All" old values associated with a the variables and equations in the model just solved being reset to default values before new solution values are brought in.

```
Transport.solveopt =0; (activates solution replace option)
Modelname.solveopt =1; (activates solution merge option -- the default)
Modelname.solveopt =2; (activates solution clear option)
```

or the option command

```
option solveopt = replace;
option solveopt = merge; (default)
option solveopt = clear;
```

The values do the following

- `replace` (0) causes the solution information for all equations appearing in the model to be completely replaced by the new model results. Variables are only replaced if they appear in the final model.
- `merge` (1) causes the solution information for all equations and variable to be merged into the existing solution information. This is the default.

© 2022 Prof. Bruce McCarl

- `clear` (2) causes the solution information for all equations appearing in the model to be completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals to be removed (set to zero).

as discussed in the Option Command and Model Attributes chapters.

3.3.4.3 Choosing a solver

GAMS offers a number of choices to solve a model and the user may switch solvers at their discretion providing they have appropriate licenses. Mechanically, there are several ways to switch

- One can just before the solve use an option command of the form

```
Option model type = solver name
```

  where **model type** is the same **model type** as that used in the **Solve** statement and solver name is the name of one of the available solvers.

```
option lp=bdmlp;
option nlp=conopt;
option MIP=cplex;
```

- One can call GAMS with the command line parameter

```
model type = solver name
```

  as in

```
gams mymodel lp=bdmlp
```

  When using the IDE this is placed in the GAMS command box in the upper right hand corner as discussed in the Running Jobs with GAMS and the GAMS IDE chapter.

- One can choose the solver using the file/options/solvers dialogue in the IDE.
- One can rerun gamsinst.exe or gamsinst.run at any time and alter the choice of default solver.
- One can generate a list of all solvers and current default solvers in the LST file using the option Subsystems.

The solvers available depends on the license file.

3.4 Model Types and Solvers

There are a number of model types and solvers in GAMS. Here we define each model type and list the solvers along with giving a cross reference of which solvers can solve which model types as of Aug 2005.

Model Types
Solver capabilities matrix
Solvers

© 2022 Prof. Bruce McCarl

3.4.1 Model Types

GAMS contains a number of model types. Each is explained here.

Linear programs (LP)
Nonlinear program (NLP)
Quadratically constrained program (QCP)
Mixed integer programming (MIP)
Relaxed mixed integer programming (RMIP)
Mixed complementarity problem (MCP)
Mixed integer nonlinear program (MINLP)
Relaxed mixed integer nonlinear program (RMINLP)
Mixed integer quadratically constrained program (MIQCP)
Relaxed mixed integer quad. constrain program (RMIQCP)
Constrained nonlinear systems (CNS)
Nonlinear programming with discontinuous derivatives (DNLP)
Mathematical program with equilibrium constraints (MPEC)
Relaxed mathematical program with equilibrium constraints (RMPEC)
Extended Mathematical Programs (EMP)

3.4.1.1 Linear programs (LP)

Mathematically, the linear programming (LP) Problem looks like:

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

where

x is a vector of variables that are continuous real numbers;
cx is the objective function;
AX α b represents the set of constraints with á being some mixture of ≤, = and ≥ operators; and
L and U are vectors of lower and upper bounds on the variables which are often 0 and infinity.

The LP problem can be solved using a number of alternative solvers in GAMS. For information on the names of the solvers that can be used on models in the LP class see the section on Solver Model type Capabilities.

3.4.1.2 Nonlinear program (NLP)

Mathematically, the nonlinear programming (NLP) Problem looks like:

Maximize or Minimize	`f(x)`
subject to	`g(x) á 0`
	`L ≤ x ≤ U`

© 2022 Prof. Bruce McCarl

where

x is a vector of variables that are continuous real numbers;
f(x) is the objective function;
g(x) represents the set of constraints;
á is some mixture of ≤, = and ≥ operators; and
L and U are vectors of lower and upper bounds on the variables.

Both f and g must be differentiable which prohibits ABS, MIN and MAX functions from appearing. (They can be included in the DNLP alternative below.) For information on the names of the solvers that can be used on models in the NLP class see the section on Solver Model type Capabilities.

Note NLP models can possibly have the nonlinear terms inactive and in such a case setting the model attribute TryLinear = 1 causes GAMS to check the model and use the default LP solver if possible.

3.4.1.3 Quadratically constrained program (QCP)

Mathematically, the quadratically constrained programming (QCP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & cx + x'Qx \\ \text{subject to} & A_i x + x' R_i x \ \alpha \ b_i \qquad \text{for all } i \\ & L \leq x \leq U \end{array}$$

where

x is a vector of variables that are continuous real numbers;
cx is the linear part of the objective function
x'Qx is the quadratic part of the objective function
$A_i x$ represents the linear part of the ith constraint;
$x' R_i x$ represents the quadratic part of the ith constraint;
b_i is the right hand side if the ith constraint;
α is some mixture of ≤, = and ≥ operators; and
L and U are vectors of lower and upper bounds on the variables.

For information on the names of the solvers that can be used on models in the QCP class see the section on Solver Model type Capabilities.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a LP problem.

3.4.1.4 Mixed integer programming (MIP)

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

© 2022 Prof. Bruce McCarl

$$
\begin{array}{lcccccccccl}
\text{Maximize or Minimize} & c_1 t & + c_2 u & + c_3 v & + c_4 w & + c_5 x & + c_6 y & + c_7 z & & \\
\text{subject to} & A_1 t & + A_2 u & + A_3 v & + A_4 w & + A_5 x & + A_6 y & + A_7 z & & \\
 & t & & & & & & & \geq 0 & \\
 & & u & & & & & & \geq 0 \text{ and} \leq L_2 \text{ and integer} & \\
 & & & v & & & & & \in (0,1) & \\
 & & & & w & & & & \in \text{SOS1} & \\
 & & & & & x & & & \in \text{SOS2} & \\
 & & & & & & y & & = 0 \text{ or} \geq L_6 & \\
 & & & & & & & z & = 0 \text{ or} \geq L_7 \text{ and integer} &
\end{array}
$$

where the

t variables are continuous real numbers
u variables can only take on integer values bounded above by L_2
v variables can only take on binary values
w variables fall into SOS1 sets exhibiting one nonzero
x variables fall into SOS2 sets exhibiting no more than two, adjacent nonzeros
y variables are semi-continuous being zero or in excess of L_6
z variables are semi-integer being zero or in excess of L_7 and integer
$c_1 t + c_2 u + c_3 v + c_4 w + c_5 x + c_6 y + c_7 z$ is the objective function,
$A_1 t + A_2 u + A_3 v + A_4 w + A_5 x + A_6 y + A_7 z \; \alpha \; b$ represents the set of constraints of various equality and inequality forms.

For information on the names of the solvers that can be used on models in the MIP class see the section on Solver Model type Capabilities. Note not all solvers cover all the cases associated with the SOS and semi variables. Thus, if you have such a problem, you should refer to the MIP capable solver manuals to discover capability. GAMS will also reject the problem if the solver cannot handle the types of variables contained.

3.4.1.5 Relaxed mixed integer programming (RMIP)

The relaxed mixed integer programming (RMIP) problem is the same as the mixed integer programming (MIP) problem in all respects except all the integer, SOS and semi restrictions are relaxed:

$$
\begin{array}{lcccccccccl}
\text{Maximize or Minimize} & c_1 t & + c_2 u & + c_3 v & + c_4 w & + c_5 x & + c_6 y & + c_7 z & & \\
\text{subject to} & A_1 t & + A_2 u & + A_3 v & + A_4 w & + A_5 x & + A_6 y & + A_7 z & & \\
 & t & & & & & & & \geq 0 & \\
 & & u & & & & & & \geq 0 \text{ and} \leq L_2 & \\
 & & & v & & & & & \geq 0 \text{ and} \leq 1 & \\
 & & & & w & & & & \geq 0 & \\
 & & & & & x & & & \geq 0 & \\
 & & & & & & y & & \geq 0 & \\
 & & & & & & & z & \geq 0 &
\end{array}
$$

This problem type is sometimes helpful when one is having trouble attaining a feasible integer solution.

© 2022 Prof. Bruce McCarl

For information on the names of the solvers that can be used on the RMIP problem class see the section on Solver Model type Capabilities.

3.4.1.6 Mixed complementarity problem (MCP)

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

Solve for Z such that

$$F_i(Z) = 0 \text{ and } L_i \leq Z_i \leq U_i$$

or

$$F_i(Z) \geq 0 \text{ and } Z_i = L_i$$

or

$$F_i(Z) \leq 0 \text{ and } Z_i = U_i$$

where

Z_i is a set of variables to be set

F(Z) is a (possibly nonlinear) function

L_i and U_i are a set of upper and lower bounds on the variables, where L_i may be –inf and U_i may be +inf

The number of Z_i variables and the number of relations $F_i(Z)$ is equal.

This problem can be written compactly as

$$F(Z) \perp L \leq Z \leq U$$

where the $\perp$ ("perp") symbol indicates pair-wise complementarity between the function F and the variable Z and its bounds. This problem does not have an objective function.

A common special case of the MCP that illustrates the basic complementarity nature of the problem.

$$F(Z) \perp Z \geq 0$$

which is often written as $F(Z) \geq 0$, $Z \geq 0$, $F(Z) * Z = 0$. In this case, the lower bound on Z implies that $F(Z) \geq 0$, with equality holding when Z is nonzero. None of this rules out the degenerate case (i.e. where a binding F(Z) has a zero complementary variable Z) but this can make for a difficult model to solve.

Another special case arises when the bounds L and U are infinite, since Z is always between its bounds. In such a case, the function F(Z) will always equal zero and the MCP then reduces to a square but in general non linear system of equations.

The MCP problem structure imposes particular requirements on specification of the equations F(Z) in the problem. This implies a couple of things.

- It is always acceptable to write the equations defining the problem with =N= relations. In this case, the bounds on F(Z) are implied by the equation/variable matching and the variable bounds.
- If the variable Z matched to F(Z) has lower bounds only (e.g if Z in GAMS terms is a positive variable) then it is acceptable to write the equation defining F with =G= relations, since this is consistent with the constraint implied by the lower bound on Z.

© 2022 Prof. Bruce McCarl

- A variable bounded only above can be matched with =L= equations.
- Free variables without bounds can be matched with =E= equations.
- If Z has both bounds then the inequality type of the F(Z) equation is indefinite and =N= must be used.
- You can write the constraints of an LP or NLP when writing down its KKT conditions as an MCP, as long as you declare the constraint inequalities and the sign restrictions on the dual multipliers correctly. By convention, the matching between GAMS equations and their .m values is correct in the sense of MCP for minimization models, while maximization models need to have the sign of the multiplier reversed.
- Difficulties Can arise with the specification of the complementary relationships when some of the variables are held at fixed values using the.fx command and the holdfixedmodel attribute is used. Namley when variables in an MCP are held fixed by a .fx command and the holdfixed command is employed This causes GAMS to drop those variables from the model passed to the solver. In turn in this case, a number of equations are also dropped to maintain a square system and difficulties can arise. Users may need to repair the specification of the complementary relationship and use of the MCPRHOLDFX option, attribute or command line parameter results in a list of the dropped equations where the difficulty may have arisen. In turn, that list helps advanced users revise the complementarity matchup within the MCP. MCPRHOLDFX can be set using an option statement, a command line parameter, or model attribute. The default value is zero resulting in no print out of the list and when set to one such a list is created.

A familiar example of complementary relationship is found in the complementary relationship between the binding nature of the constraints in a problem and the associated dual multipliers: if a constraint is non-binding its dual multiplier must be zero (i.e. at bound) while if a dual multiplier is nonzero the associated constraint must be binding. In fact, the Kuhn Karush Tucker conditions or theoretical conditions that characterize many model types in economics (especially in the general equilibrium class) and engineering can be expressed as an MCP. See *http://www.neos-guide.org/content/complementarity-problems* for more discussion of this class of problems.

Complementarity problems are easily specified in GAMS. The only additional requirement is the definition of complementarity pairs as discussed in the Variables, Equations, Models and Solves chapter or in the NLP and MCP Model Types chapter.

For information on the names of the solvers that can be used on the MCP problem class see the section on Solver Model type Capabilities.

3.4.1.7 Mixed integer nonlinear program (MINLP)

Mathematically, the mixed integer nonlinear programming (MINLP) problem looks like:

Maximize or Minimize	`f(x) + d(y)`
subject to	`g(x) + h(y) α 0`
	`L ≤ x ≤ U`
	`y = {0,1,2,..}`

where

x is a vector of variables that are continuous real numbers;
f(x) + d(y) is the objective function,
g(x) + h(y) represents the set of constraints.

© 2022 Prof. Bruce McCarl

α is some mixture of ≤ , = and ≥ operators.
L and U are vectors of lower and upper bounds on the variables.

For information on the names of the solvers that can be used on the MINLP problem class see the section on Solver Model type Capabilities. Note SOS and semi variables can also be accommodated by some solvers as listed above in the MIP section.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a MIP problem.

3.4.1.8 Relaxed mixed integer nonlinear program (RMINLP)

The relaxed mixed integer nonlinear programming (RMINLP) problem is the same as the mixed integer nonlinear programming (MINLP) problem in all respects except the integer restriction on y is relaxed:

Maximize or Minimize	`f(x) + d(y)`
subject to	`g(x) + H(y) α 0`
	`L ≤ x ≤ U`
	`y ≥ 0`

This problem type is sometimes helpful when one is having trouble attaining a feasible integer solution. For information on the names of the solvers that can be used on the RMINLP problem class see the section on Solver Model type Capabilities.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a RMIP problem.

3.4.1.9 Mixed integer quadratically constrained program (MIQCP)

Mathematically, the mixed integer quadratically constrained programming (MIQCP) problem looks like:

Maximize or Minimize	`cx + x'Q x`	
subject to	$A_i x + x' R_i x \; \alpha \; b_i$	for all i
	`L ≤ x ≤ U`	
	`y is a subset of x restricted to equal {0,1,2,..}`	

where

x is a vector of variables that contains continuous and integer members;
y is a subset of x that contains integer members;
cx is the linear part of the objective function
x'Qx is the quadratic part of the objective function
$A_i x$ represents the linear part of the ith constraint;
$x' R_i x$ represents the quadratic part of the ith constraint;
b_i is the right hand side if the ith constraint;
α is some mixture of ≤, = and ≥ operators; and
L and U are vectors of lower and upper bounds on the variables.

© 2022 Prof. Bruce McCarl

For information on the names of the solvers that can be used on models in the MIQCP class see the section on Solver Model type Capabilities.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a MIP problem.

3.4.1.10 Relaxed mixed integer quad. constrain program (RMIQCP)

Mathematically, the relaxed mixed integer quadratically constrained programming (RMIQCP) problem looks like:

Maximize or Minimize	$cx + x'Q\ x$
subject to	$A_i x + x' R_i x\ \alpha\ b_i$ for all i
	$L \leq x \leq U$
	y is a subset of x relaxed from integer to continuous

where

x is a vector of variables that contains continuous and integer members;
y is a subset of x that contains relaxed integer members;
cx is the linear part of the objective function
x'Qx is the quadratic part of the objective function
$A_i x$ represents the linear part of the ith constraint;

$x' R_i x$ represents the quadratic part of the ith constraint;

b_i is the right hand side if the ith constraint;

α is some mixture of ≤, = and ≥ operators; and
L and U are vectors of lower and upper bounds on the variables.

For information on the names of the solvers that can be used on models in the RMIQCP class see the section on Solver Model type Capabilities.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a RMIP problem.

3.4.1.11 Constrained nonlinear systems (CNS)

Mathematically, a constrained nonlinear system (CNS) model looks like:

find	x
subject to	$F(x) = 0$
	$L \leq x \leq U$
	$G(x) \leq b$

where

x is a set of variables
F is a set of nonlinear equations.

© 2022 Prof. Bruce McCarl

In addition the number of equations and the number of unknown variables x need to be 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.

The CNS model is a generalization of a problem form involving solve for x over a system of equations with one equation present for each x (a square system) like 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 are generated by the solver (e.g. singular at solution, locally unique solution), and
- A potential reduction in solution times, by taking better advantage of the model properties.

For information on the names of the solvers that can be used on the CNS problem class see the section on Solver Model type Capabilities.

3.4.1.12 Mathematical program with equilibrium constraints (MPEC)

Mathematically, the mathematical program with equilibrium constraints (MPEC) looks like:

$$
\begin{array}{ll}
\text{Maximize or Minimize} & f(x,y) \\
\text{subject to} & g(x,y) \; \alpha \; 0 \\
 & F(x,y) \perp L_y \leq y \leq U_y \\
 & L_x \leq x \leq U_x
\end{array}
$$

where

x and y are vectors of variables that are continuous or discrete where the variables x are often called the state variables or upper-level variables, while the variables y are called the control or lower-level variables.
f(x,y) is the objective function.
g(x,y) represents the set of constraints; in some cases, they can only involve the state variables x.
F(x,y) and the bounds L_y and U_y define the equilibrium constraints.

Note: If x is fixed, then F(x,y) and the bounds L_y and U_y define an MCP. 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, there is only an experimental MPEC solver included in the GAMS distribution. For more details see *http://gamsworld.org/mpec/index.htm* and *http://www.neos-guide.org/content/complementarity-problems*.

© 2022 Prof. Bruce McCarl

3.4.1.12.1 Nonlinear programming with discontinuous derivatives (DNLP)

Mathematically, the nonlinear programming with discontinuous derivatives (DNLP) problem looks like:

Maximize or Minimize	f(x)
Subject to	g(x) α 0
	L ≤ x ≤ U

where

x is a vector of variables that are continuous real numbers,
f(x) is the objective function,
α is a set of inequality and equality operators
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). However one should note that the solvers may have problems when dealing with the discontinuities due to the fact that the solvers are really NLP solvers that are used on DNLPs and the optimality conditions plus the reliance on derivatives may be problematic.

Use of BARON, CBC and DICOPT may alleviate this problem. For information on the names of the solvers that can be used on the DNLP problem class see the section on Solver Model type Capabilities.

Use of the model attribute TryLinear causes GAMS to see if the problem can be solved as a LP problem.

3.4.1.13 Relaxed mathematical program with equilibrium constraints (RMPEC)

The relaxed mathematical program with equilibrium constraints (RMPEC) is exactly like the MPEC problem but any integer variables that are present are relaxed to be continuous.

This problem type is sometimes helpful when one is having trouble attaining a feasible integer solution.

3.4.1.14 Extended Mathematical Programs (EMP)

EMP (Extended Mathematical Programming) is an (experimental) framework for automated mathematical programming reformulations. Models which currently cannot be solved reliably are reformulated into models of established mathematical programming classes in order to use advanced solver technology.

Today EMP supports
Bilevel Programs
Disjunctive Programs
Extended Nonlinear Programs
Embedded Complementarity Systems
Variational Inequalities

The reformulation is done by the solver JAMS which is the only solver that is capable to handle EMP models.

© 2022 Prof. Bruce McCarl

Examples showing how to use the EMP framework and the solvers JAMS and DE are made available through the
GAMS EMP Library which is included in the GAMS Distribution.

EMP has been developed jointly by Michael Ferris of UW-Madison, Ignacio Grossmann of Carnegie Mellon University and GAMS Development Corporation. DE and JAMS come free of charge with any licensed GAMS system but need a subsolver to solve the generated models.

Discussion on EMP appears in this document

There is a stochastic extension called EMPSP.

3.4.2 Solver capabilities matrix

The solvers capabilities matrix (as obtained and explained in the Using GAMS and the GAMS IDE chapter) from the IDE and GAMS 22.6 is below. Entries in the cells show where a solver can solve a problem with an X indicating the default solver and a – indicating a non solving utility. The full and Demo on the left hand side reflect solver licensing status. The subsystems option command also lists this information. An updated version of this matrix can be obtained from the GAMSIDE or using the subsystems option.

See model types and platforms supported by each solver.

3.4.3 Solvers

GAMS Corporation, in general, does not develop software to solve models. Rather they develop procedures linking to software developed by others and then GAMS automatically links to that software whenever a model is to be solved. A variety of third party software has been interfaced with GAMS and can be licensed through the GAMS Corporation. In this section I list the third party software available as of GAMS version 20.5 giving some of its broad characteristics.

We should note that the solver manuals are the true reference source for each solver and we provide hyperlinks to the latest version(s) of that documentation available. Before doing this we briefly discuss licensing and the reasons for and conventions behind the way alternative versions of the same software item are distributed.

General notes on solver licensing
General notes on solver versions
Available solvers
Choosing a solver

3.4.3.1 General notes on solver licensing

A wide variety of add on solvers are available to be used with GAMS almost all of which are the product of third party software development groups. The base version of GAMS comes with

© 2022 Prof. Bruce McCarl

- A fully implemented version of the BDMLP software which can be used to solve LP, MIP and RMIP model types.
- A version of MILES that can be used to solve MCP model types.
- Access to the COIN solvers that can solve all of the model types.
- General-purpose utilities such as CONVERT and GAMSCHK.
- Other solvers in demonstration (demo) mode wherein the maximum problem size that can be fit into the solver is limited.
- The BDMLP, MILES, COIN and other solvers under a demo license exhibit lesser capability and or reduced speed or reliability relative to the available additional solvers.
- Fully capable versions of the other solvers can be made available by expanding ones GAMS license by arrangement through GAMS Corporation at *mailto:sales@gams.com*.
- Certain solvers are available without charge to academic users through academic initiatives by the solver developers as discussed in https://support.gams.com/solver:academic_programs_by_solver_partners
- When a license error is encountered the system parameters LicenseLevel and LicenseLevelText and the functions LicenseLevel and LicenseStatust can be used to recover messages and indicators.
- A multiplatform license file was introduced as of distribution 23.6 which will work on all platforms supported by GAMS.

3.4.3.2 General notes on solver versions

Cases exist where GAMS distributes more than one version of the third party solvers. This occurs for several reasons.

- Alternative solver versions may be available that exploit specialized computer hardware capability. For example, there are parallel processor versions of CPLEX and XA.
- Alternative solver versions may be available that incorporate enhanced capabilities, but also involve a higher licensing fee. For example, one may license CPLEX solely for simplex based linear program solutions or gain access to expanded capability versions which contains an interior point algorithm and solve mixed integer programming models.
- Alternative solver versions may be available that are capable of solving different problem classes. For example, there are versions of PATH for solving mixed complementarity programs. But there are also versions that are applicable to nonlinear programming models. Similarly, there are versions of OSL specialized for stochastic programming model extensions.
- Alternative software versions may be available if there are either beta test versions being distributed or experience has shown there are cases where older versions perform better in

© 2022 Prof. Bruce McCarl

some cases.

In the face of all of these possible alternative versions a consistent naming convention has been adopted for a few of the solvers. The following rules apply:

- The base name of the solver does not change (e.g. CPLEX, OSL, CONOPT, ...) and refers to the current production version.
- Past and beta versions have names derived from the base name generally with a number appended or letters.
- If you select the base name of a solver you will get the most recent production version of the solver.

For example, in the release current in March 2002 the system contains CONOPT versions CONOPT1 (past), CONOPT2 (production), and CONOPT3 (future). When a solver with the base name CONOPT is identified then GAMS uses the current production version or CONOPT2.

3.4.3.3 Available solvers

Now we list the available solvers including a little bit on their capabilities and pedigree as well as a hyperlink to available documentation.

© 2022 Prof. Bruce McCarl

ALPHAECP
BARON
BONMIN
CBC
CONOPT
CONVERT
CPLEX
DE
DECISC
DICOPT
EMP
EMPSP
EXAMINER
GAMSCHK
GLOMIQO
GUROBI
GUSS
IPOPT
JAMS
KESTREL
KNITRO
LGO
LINDOGLOBAL
MILES
MINOS
MOSEK
MPS2GMS
MPSGE
MSNLP
NLPEC
OSICPLEX
OSIGUROBI
OSIMOSEK
OSIXPRESS
OSISOPLEX
PATH
SCENRED
SCIP
SNOPT
SOPLEX
XA
XPRESS

3.4.3.3.1 ALPHAECP

ALPHAECP solves mixed integer non-linear problems. It is an implementation of the Extended Cutting Plane method by Tapio Westerlund and Toni Lastusilta from Abo Akademi University, Finland. It use requires the presence of a licensed MIP solver. The solver documentation is at **alphaecp**.

The ECP method is an extension of Kelley's cutting plane method which was originally given for convex NLP problems (Kelley, 1960). The method requires only the solution of a MIP sub problem in each iteration. The MIP sub problems may be solved to optimality, but can also be solved to feasibility or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. Futher information about the underlying algorithm can be found in Westerlund T. and Pörn R. (2002). Solving Pseudo-Convex Mixed Integer Optimization Problems by Cutting Plane Techniques. Optimization and Engngineering, 3. 253-280.

3.4.3.3.2 ANTIGONE

The solver ANTIGONE (Algorithms for coNTinuous / Integer Global Optimization of Nonlinear

© 2022 Prof. Bruce McCarl

Equations) is a computational framework for deterministic global optimization of nonconvex MINLP. ANTIGONE performs equivalently to GloMIQO when all nonlinearities in MINLP are quadratic. ANTIGONE has been developed by Computer-Aided Systems Laboratory at Princeton University; it was completed in collaboration with the Centre for Process Systems Engineering at Imperial College.

GAMS/ANTIGONE is available for the 32-bit and 64-bit versions of Windows and Linux.

Usage of GAMS/ANTIGONE requires the presence of a GAMS/CPLEX license and either a GAMS/CONOPT or a GAMS/SNOPT license.

ANTIGONE can solve MINLP problems plus simplifications thereof.

It is documented commercial part of the solver manual.

3.4.3.3.3 BARON

BARON is a solver that is designed to find globally optimal solutions for nonconvex optimization model types. Baron was developed by N. Sahinidis, M. Tawarmalani and associates at the University of Illinois-Urbana-Champaign. Purely continuous, purely integer, and mixed-integer nonlinear model types can be solved. The BARON acronym stands for the branch and reduce optimization navigator. It derives its name from its combining of constraint propagation, interval analysis, and duality approaches in a problem reduction arsenal along with enhanced branch and bound concepts. BARON uses these procedures to searching for global solutions in the face of the non convex hills and valleys a problem structure may exhibit.

BARON encompasses:

- A general purpose solver for optimization problems with nonlinear constraints and/or integer variables.
- Fast specialized solvers for linearly constrained problems.
- Capability to solve LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP model types.

The GAMS solver manual for BARON is baron and a more general manual is on *http://www.minlp.com/downloads/docs/baron manual.pdf* and a more general discussion of the whole area appears on *http://www.minlplib.org/*.

3.4.3.3.4 BONMIN/BONMINH

BONMIN: Bonmin (Basic Open-source Nonlinear Mixed Integer programming) 0.9 is an open-source solver for mixedinteger nonlinear programming (MINLPs), whereof some parts are still experimental. The code is developed in a joint project of IBM and the Carnegie Mellon University. The COIN-OR project leader for Bonmin is Pierre Bonami.

Bonmin implements five different algorithms:

- B-BB: a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on variables

© 2022 Prof. Bruce McCarl

- B-OA: an outer-approximation based decomposition algorithm

- B-QG: an outer-approximation based branch-and-cut algorithm (by Queseda and Grossmann)

- B-Hyb: a hybrid outer-approximation/nonlinear programming based branch-and-cut algorithm (default)

- B-Ecp: an ECP cuts based branch-and-cut algorithm a la FilMINT

The algorithms are exact when the problem is convex, otherwise they are heuristics.

An in core link exists and has the solver name BONMIND.

BONMIN was previously named COINBONMIN.

BONMINH is a commerically licensed version that makes available use of the linear solvers MA27, MA57, and MA86 from the Harwell Subroutines Library (HSL) in IPOPT.

3.4.3.3.5 CBC

CBC (formerly COINCBC) is a free solver for mixed integer nonlinear programs that arises from the *COIN-OR* project (Computational Infrastructure - Operations Research). The COIN-OR project releases open-source software for the operations research community. The GAMS/COIN-OR Link is available in source and free of charge with any licensed GAMS system. The CBC (COIN Branch and Cut) solver was developed under the leadership of John Forest at IBM and is documented in a manual authored by John Forrest and Robin Lougee-Heimer at http://www.coin-or.org/Cbc/cbcuserguide.html. A very brief GAMS related solver manual is available with more GAMS implementation related details including procedures to change the integer seeking branch and bound and cuts strategies at https://www.gams.com/latest/docs/S_CBC.html.

An in core link exists and has the solver name CBCD.

© 2022 Prof. Bruce McCarl

3.4.3.3.6 CSDP

CSDP, is a solver that that implements a predictor corrector variant of the semidefinite programming algorithm of Helmberg, Rendl, Vanderbei, and Wolkowicz. Detailed descriptions of CSDP and its parallel version can be found in the following papers.

- B. Borchers. CSDP, A C Library for Semidefinite Programming. Optimization Methods and Software 11(1):613-623, 1999

3.4.3.3.7 CONOPT

CONOPT is a solver for large-scale nonlinear optimization (NLP) developed and maintained by A. Drud, ARKI Consulting & Development. CONOPT is a feasible path solver based on the generalized reduced gradient (GRG) method. CONOPT contains extensions to the GRG method such as a special phase 0, linear mode iterations, and a sequential linear programming component. CONOPT can solve the LP, RMIP, NLP, CNS, DNLP, and RMINLP model types. The solver manual is in `CONOPT` and other material can be found on *http://www.conopt.com/*.

An alternative version of CONOPT is distributed that is designed to speedup GAMS to solver communications and is called CONOPTD.

3.4.3.3.7.1 CONOPTD

CONOPTD is an experimental versions of CONOPT that passes information from GAMS to CONOPT in core without use of temporary files. It is designed to speed up GAMS to solver communications. It's use also requires use of the command `modelname.solvelink=5;` before the solve statement.

3.4.3.3.8 CONVERT

CONVERT developed by GAMS Corporation transforms a GAMS model instance into a format used by other modeling and solutions systems. CONVERT is designed to achieve three aims:

- Permit users of GAMS to convert a confidential model into a GAMS solvable scalar form with very little identifying its structure so it can be given to a group for numerical investigation (i.e to have someone else help with a solution problem while maintaining confidentiality).
- Give a path to solving with other solvers that may not be available in GAMS to test performance.
- Give a way of sharing test problems.

Currently, CONVERT can translate GAMS models into a AlphaECP, AMPL, AmplNLC, BARON, CoinFML, CplexLP, CplexMPS, Dict, FixedMPS, GAMS Scalar format, LAGO, LGO, LindoMPI, LINGO, MINOPT, NLP2MCP or ViennaDag type of problem. In addition Jacobian creates a GDX file containing the basic model data (matrix, initial point, evaluation of constraints at initial point and bounds).

The translator creates a "scalar model" which consists of

- A model without sets or indexed parameters in the scalar models, that is one that does not exploit the more advanced characteristics of any modeling system and is easily transformable.
- A model with a new set of individual variables depicting each variable in the GAMS model

© 2022 Prof. Bruce McCarl

ending up with potentially 3 variable classes for the positive, integer, and binary variables each numbered sequentially (i.e. all positive GAMS variables are mapped into n single variables X1 - Xn thus if we have transport(i,j) and manufacture(k) we would have a set of unindexed scalar variables X1, X2, ... with i*j+k cases.),

- A model with individual equations depicting each variable in the GAMS model (ie all GAMS equations are mapped into m constraints E1, E2, ... Em thus if we have demand(j) and resources(r,k) we would have a new equations E1, E2,... with j+r*k cases),
- The symbolic form of these equations.
- Bounds or starting point values.

Models of the types LP, MIP, RMIP, NLP, MCP, MPEC, CNS, DNLP, RMINLP and MINLP can be converted. Convert contains the program writeup.

CONVERTD is a faster, in core, experimental version.

3.4.3.3.8.1 CONVERTD

CONVERTD is an experimental version of CONVERT that passes information from GAMS to CONVERT in core without use of temporary files. It is designed to speed up GAMS to solver communications.

3.4.3.3.9 CPLEX

CPLEX is a solver for linear, mixed-integer and quadratic programming problems developed by ILOG (http://www.ilog.com/products/cplex/). CPLEX contains a primal simplex algorithm, a dual simplex algorithm, a network optimizer, an interior point barrier algorithm, a mixed integer algorithm and a quadratic capability. CPLEX also contains an infeasibility finder. For problems with integer variables, CPLEX uses a branch and bound algorithm (with cuts) and supports specially ordered set variables SOS1, SOS2 as well as semi-continuous and semi-integer variables. Base CPLEX solves LP and RMIP model types. Additional capabilities of CPLEX can be licensed involving Barrier, MIP and QCP capability. The write-up for CPLEX is on cplex

A bare bones free version is available as OSICPLEX.

3.4.3.3.9.1 CPLEXD

CPLEXD is an experimental versions of CPLEX that passes information from GAMS to CPLEX in core without use of temporary files. It is designed to speed up GAMS to solver communications. It's use also requires use of the command `modelname.solvelink=5;` before the solve statement. It currently does not have all of the capabilities of the full CPLEX version.

3.4.3.3.10 DE

DE is a "solver" that accepts EMP models that have been annotated with information about uncertainty. The DE “solver” reformulates the stochastic model into the extensive form equivalent with implicit non-anticipativity constraints. The reformulated model is solved with any of the regular GAMS solvers. All optimization model types (LP, MIP, QCP, MIQCP, NLP, DNLP, and

© 2022 Prof. Bruce McCarl

MINLP) are accepted. This is written up in the EMP SP document at empsp which is also in the help files of the GAMSIDE under solvers and the title **Stochastic Programming (SP) with EMP.**

3.4.3.3.11 DEA

DEA is a older contributed solver that dealt with linear and mixed integer Data Envelopment Analysis (DEA) programs, as well as other linear, mixed integer and simple quadratic slice models. It has been replaced by GUSS as explained at http://www.gams.com/contrib/gamsdea/dea.htm. .

3.4.3.3.12 DECIS/DECISC/DECISM

DECIS is a system developed by G. Infanger of Stanford University and Infanger Investment Technology for solving stochastic linear programs. Such programs include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to have known probability distributions. It employs Benders decomposition and Monte Carlo sampling techniques. DECIS includes a variety of solution strategies, such as solving the universe problem, the expected value problem, Monte Carlo sampling within the Benders decomposition algorithm, and Monte Carlo presampling. For solving linear and nonlinear programs (master and subproblems arising from the decomposition)

DECISC interfaces with CPLEX and thus requires its presence.

DECISM interfaces with MINOS and thus requires its presence.

The manual is in **DECIS**. DECISC can be used to solve LP model types.

3.4.3.3.13 DICOPT

DICOPT solves mixed-integer nonlinear programming (MINLP) model types. It was developed by J. Viswanathan and I. Grossmann at Carnegie Mellon University. The models solved involve linear binary or integer variables and linear and nonlinear continuous variables. Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum.

DICOPT implements extensions of the outer-approximation algorithm for the equality relaxation strategy. The DICOPT solution approach involves solving a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP or MIP solver that runs under GAMS. This allows one to match the best algorithms to the problem at hand and guarantees that enhancements in the NLP and MIP solvers are exploited. It also requires licenses to such solvers. DICOPT solves MINLP model types. The solver manual is on **DICOPT**.

3.4.3.3.14 EMP

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematical programming reformulations. The idea behind EMP is that new upcoming types of models which currently cannot be solved reliably are reformulated into models of established math

© 2022 Prof. Bruce McCarl

programming classes in order to use mature solver technology. At this stage, EMP supports the modeling of Bilevel Programs, Variational Inequalities, Disjunctive Programs, Extended Nonlinear Programs and Embedded Complementarity Systems, but additional features are being added regularly.

It is documented here.

3.4.3.3.15 EMPSP

This solver does Stochastic Programming (SP) with EMP and is an extension of GAMS EMP. It builds a stochastic model based on a users deterministic model following a users definition of which model parameters to be uncertain. In turn the GAMS EMP replaces these uncertain parameters with distributions. The distribution of the random parameters is controlled by the user. .

The documentation and examples are in empsp .

3.4.3.3.16 EXAMINER

EXAMINER is not a solver per se but rather a tool that lets the user examine the merit of a solution returned by a solver. In short, it checks to see if solutions are satisfactory. Namely given a solution point reported as optimal by a solver the EXAMINER software investigates primal feasibility, dual feasibility, and optimality. It is mainly designed to help in solver development, testing, and debugging but also allows comparison of solutions across solvers or solver settings (e.g. alternative optimality tolerances and optimality criteria). EXAMINER is distributed with all systems and can run in DEMO mode. The solver manual is on **examiner**.

3.4.3.3.17 GAMSCHK

GAMSCHK is a program developed at Texas A&M University that is designed to aid users who wish to examine empirical GAMS models for possible flaws. GAMSCHK will:

- List coefficients for user selected equations and/or variables.
- List the characteristics of selected groups of variables and/or equations.
- List the characteristics of equation and variable blocks.
- Examine a GAMS model to see whether any variables and equations contain specification errors.
- Generate schematics depicting the characteristics of coefficients by variable and equation blocks.
- Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude.
- Reconstruct the reduced cost of variables and the activity within equations after a model solution.
- Help resolve problems with unbounded or infeasible models.

GAMSCHK does not actually solve the problem but rather requires other solvers to solve the problem then saves the basis information. It will work with LP, MIP, RMIP, NLP, MCP, DNLP, RMINLP, and MINLP model types. The manual is on **GAMSCHK**.

© 2022 Prof. Bruce McCarl

3.4.3.3.18 GLOMIQO

GloMIQO (Global Mixed-Integer Quadratic Optimizer) is a numerical solver that solves mixed-integer quadratically-constrained quadratic programs to ε-global optimality. It was developed by R. Misener and C. A. Floudas.

It is discussed at http://ares.tamu.edu/GloMIQO/ and the GAMS version is documented in GloMIQO solver manual.

GloMIQO is available through Princeton University and in a beta version in GAMS as of release 23.8.

3.4.3.3.19 GUROBI

GUROBI is a soon to be fully released solver that provides state-of-the-art simplex-based linear programming (LP) and mixed-integer programming (MIP) capability. The GUROBI MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths. The Gurobi solver is available for the 32-bit and 64-bit versions of Windows and Linux. Details on GYROBI can be found at http://www.gurobi.com/ .

A bare bones, free version of GUROBI is available as COINGUROBI.

3.4.3.3.20 GUSS

GUSS is a GAMS facility that permits solution of a set of scenarios for a GAMS model modifying data to run each scenario. GUSS allows the collection of models to be solved in a single pass without needing repeated solves or a LOOP over multiple solves. GUSS is not really a solver but rather organizes and passes data to the other gams solvers for most model types. This is all done in a faster fashion than say when using multiple solves through the GAMS Loop command and is much faster for small models.

In particular GUSS runs the model repeatedly over user specified data for model parameters that collectively define alternative scenarios to be run. In doing this it repeatedly updates the base model with the altered scenario data, then solves the updated model for that scenario and saves user chosen results for each scenario.

© 2022 Prof. Bruce McCarl

GUSS was developed by Michael R. Bussieck, Michael C. Ferris, and Timo Lohmann. It is documented in http://www.gams.com/modlib/adddocs/gusspaper.pdf and in the GUSS section of the solver manual.

GUSS is available in all versions of GAMS starting with release 23.7.

Use of GUSS for an existing model requires six steps

1. Definition of scenarios to run
2. Definition of parameters holding scenario specific data for the items in the model that to be changed
3. Definition of parameters that will hold scenario specific model results for the items that the user wished to save
4. Definition of a set that tells GUSS the scenarios to run, data to change and results to save
5. Modification of the solve statement to identify that scenarios will be run
6. Development of code to report the scenario results

Each will be covered below.in the context of the model in risk.gms which originally solved a model repeatedly for different risk aversion parameters using the code below

```
loop (raps,rap=riskaver(raps);
            solve evportfol using nlp maximizing obj ;
            var = sum(stock, sum(stocks,
                invest.l(stock)*covar(stock,stocks)
*invest.l(stocks))) ;
            output("rap",raps)=rap;
            output(stocks,raps)=invest.l(stocks);
            output("obj",raps)=obj.l;);
```

We will now discuss the 6 steps of set up for an example based on the risk model which we call GUSSRISK.gms.

1. Definition of scenarios to run

The first step in the procedure is to establish a set that covers the scenarios that will be run. For the risk.gms example the alternative runs are ones for different risk aversion parameters controlled by the set RAPS. So in GUSSRISK.gms we will define a set of names for risk aversion parameters and give it the name **RAPSCENARIOS** in a set statement as follows

```
SET RAPSCENARIOS    RISK AVERSION PARAMETERS /R0*R25/
```

2. Definition of parameters holding scenario specific data

The second step in the procedure is to establish scenario dependent values for the model data items that will be changed across the scenarios that will be run. For the GUSSRISK.gms example we wish to use scenario dependent risk aversion parameters

© 2022 Prof. Bruce McCarl

defined over the set **RAPSCENARIOS**. We do this in a parameter statement as follows

```
PARAMETER RISKAVER(RAPSCENARIOS) RISK AVERSION
COEFICIENT BY RISK AVERSION PARAMETER
                /R0   0.0000000001,  R1   0.00025,  R2
0.00050,  R3   0.00075,
                R4   0.00100,  R5   0.00150,  R6
0.00200,  R7   0.00300,
                R8   0.00500,  R9   0.01000,  R10
0.01100,  R11  0.01250,
                R12  0.01500,  R13  0.02500,  R14
0.05000,  R15  0.10000,
                R16  0.30000,  R17  0.50000,  R18
1.00000,  R19  2.50000,
                R20  5.00000,  R21  10.0000,  R22  15.
  ,  R23  20.
                R24  40.    ,  R25  80./           ;
```

In general, the parameter storing the data has to have the scenario set in its first index position. Thus, if one is altering a scalar like RAP in the GUSSRISK.gms example, we define a one dimensional parameter over the scenario set in this case **RISKAVER (RAPSCENARIOS).**

When the scenario analysis involves modifying a more complex parameter like one named **modelparam(i,j,k)** one would define a new parameter with a structure like **newmodelparam(scenarioset,i,j,k)** where **scenarioset** is the set of scenarios that will be handled by GUSS and the i,j,k are the original set definitions in the parameter to be changed..

Note we could have changed more than one parameter but here this is all we will modify. For a more complex example see gussexample1.gms.

3. Definition of parameters to hold scenario specific model results

The third step in the procedure is to establish scenario dependent repositories where the scenario dependent model solution results will be stored. For the GUSSRISK.gms example, we will store the levels of investment, the objective function value and the available funds shadow price. A pararamer will be defined to hold each of these items and those parameters must have the named **scenario set in the first index position** plus the other index positions need to contain the full dimension of the solution item to store. In the GUSSRISK.gms example the scenario set is **RAPSCENARIOS** and we define the parameters as follows

```
PARAMETER
```

© 2022 Prof. Bruce McCarl

```
     STOCKOUTPUT(RAPSCENARIOS,STOCKS) RESULTS FOR
INVEST with VARYING RAP
     OBJLEVEL(RAPSCENARIOS) OBJECTIVE FUNCTION WITH
VARYING RAP
     INVESTAVshadow(RAPSCENARIOS) FUNDS SHADOW PRICE
WITH VARYING RAP
;
```

One can also specify a parameter to hold solution status information relative to the solutions for each scenario. In that case we specify the nature of the information we want and the array name to hold the information. Again this must have the scenario set name in the first index position. In the GUSSRISK.gms example this involves the statements

```
Set       modelattrib model solution information to collect /
modelstat, solvestat, objval /;
PARAMETER solutionstatus(RAPSCENARIOS, modelattrib) Place to
store Solution status reporting
*             assign initial values
             / #RAPSCENARIOS.(ModelStat na, SolveStat na,
ObjVal na) /;
```

Where

- the first line defines a set that contains the names of the model attributes to store using in this case the attributes for
  - model solution status (modelstat with an explanation of the possible numerical values given here)
  - solver solution status (solvestat with an explanation of the possible numerical values given here)
  - the optimal value of the objective function (objval)
  - Note more items can be stored and are domusd, iterusd, objest, nodusd, numinfes, robj, suminfes, maxinfes, and meaninfes as mostly defined in the list of model attributes

© 2022 Prof. Bruce McCarl

here

- the second line defines the parameter in which the values are to be stored
- The fourth line initializes all values to na and if the solves fail then those values will remain. It also uses the # operator to address all set elements as discussed here.

4. Definition of a set that tells GUSS what to do

The fourth step in the procedure is to establish a three dimensional tuple that tells GUSS what you wish to do.

The tuple contains

- the name of the set defining the scenarios
- the names of model parameters to be changed and the name of the parameters where the scenario dependent data are stored
- the names of model solution parameters to be saved along with identification of the type of the solution information to save and the name of a the place where to save it.
- the names of a parameter with options to pass to GUSS along with the name of the place to store solution attributes

For the GUSSRISK.gms example the statement is as follows.

```
set GUSSdict   / RAPSCENARIOS.scenario   . ''
                 rap           .param      .RISKAVER
                 INVEST        .level      .STOCKOUTPUT
                 OBJ           .level      .OBJLEVEL
                 INVESTAV      .marginal
.INVESTAVshadow
               /;
```

Here the named set is GUSSDICT and

- the first line identifies the name of the set defining the scenarios (**RAPSCENARIOS**) and associates it with the word **SCENARIO** and a third entry of **''**
- the second line identifies the name of the data element in the model (**rap**) to be changed in running the scenarios and associates it with the word **param** and an entry telling where the alternative values are held (**RISKAVER**)
- the third line identifies the name of a solution output to store (**INVEST**), its nature (a **level** or **INVEST.L** in this case) and the place to store it (**STOCKOUTPUT**)
- the fourth line identifies the name of a solution output to store (**OBJ**), its nature (a **level** or **OBJ.L** in this case) and the place to

© 2022 Prof. Bruce McCarl

store it (**OBJLEVEL**)

- the fifth line identifies the name of a solution output to store (**INVESTAV**), its nature (a **marginal** or **INVESTAV.M** in this case) and the place to store it (**INVESTAVshadow**)

Note the key words that can be used in the second tuple position are

param	Indicating this is an item that provides scenario data for a model parameter that will be altered
lower	Indicating this is an item that provides alternative lower bounds for model variables or equation RHS's that will be changed
upper	Indicating this is an item that provides alternative upper bounds for model variables or equation RHS's that will be changed
fixed	Indicating this is an item that provides alternative fixed bounds for model variables or RHS's that will be changed
level	Indicating this is an item that will be used to store solution levels for model variables or equations
Marginal	Indicating this is an item that will be used to store solution marginals for model variables or equations
opt	Indicating the parameter holding GUSS options to use and where to store model solution attributes

One may also modify multiple input parameters as in the tuple specified in gussexample1.gms where parameters a and b take on multiple values.

```
set dict   / scenariostorun.scenario  .''
             gussoptions   .opt       .solutionstatus
             a             .param     .newsupply
             b             .param     .newdemand
             x             .level     .resultantx
           /
```

If one uses the opt command one also needs to specify a parameter that holds options for GUSS using syntax like

```
parameter gussoptions  options to use in running GUSS
                                / UpdateType 1, Optfile 1 /
```

The available options are discussed in the solver manual in the GUSS section with the most important ones involving option files to use controlling the option files to use for the first and subsequent solves, the amount of output in the LOG file, the way the data update is performed and the type of solution point to restart from.

© 2022 Prof. Bruce McCarl

5. Modification of the solve statement to identify that scenarios will be run

The fifth step involves altering the solve statement so it both knows that GUSS is to be used plus an identification of the name of the tuple that passes instructions on what GUSS needs to do.

The format of this in the GUSSRISK.gms example is

```
SOLVE EVPORTFOL USING NLP MAXIMIZING OBJ SCENARIO
GUSSDICT ;
```

where the solve statement is of the conventional form with the addition of the key word **SCENARIO** and name of the tuple from step 4 that tells GUSS what to do. In this case the name of that tuple is **GUSSDICT.**

In the gussexample1.gms case the solve statement is

```
Solve transport using lp minimizing z scenario dict;
```

where again we have the addition of the key word **SCENARIO** and **DICT** is the name of the tuple that tells GUSS what to do.

6. Development of code to report the scenario results

The sixth step involves implementing post solution instructions to report the scenario dependent family of solutions to the user. This is done either directly through a display or through calculation of tables and inclusion in output through display, put files or passing to other programs as discussed elsewhere in this guide.

In the example in gussexample1.gms we simply display the array

```
option resultantx:0:1:2;
display resultantx,solutionstatus;
```

In GUSSRISK.gms we run through a report writing loop placing the scenario dependent solution information into the model variable levels and shadow prices and then build a report table

```
 PARAMETER OUTPUT(*,rapscenarios);
 LOOP (RAPSCENARIOS,RAP=RISKAVER(RAPSCENARIOS);
*           LOAD IN SOLUTION INFORMATION
            INVEST.L(STOCKS)=STOCKOUTPUT
(RAPSCENARIOS,STOCKS);
            OBJ.L=OBJLEVEL(RAPSCENARIOS);
            INVESTAV.m=INVESTAVshadow(RAPSCENARIOS);
```

© 2022 Prof. Bruce McCarl

```
*           COMPUTE SOME ITEMS
            INVESTAV.L=SUM(STOCKS,INVEST.L(STOCKS));
            VAR = SUM(STOCK, SUM(STOCKS,
                INVEST.L(STOCK)*COVAR(STOCK,STOCKS)
*INVEST.L(STOCKS))) ;
            OUTPUT("RAP",rapscenarios)=RAP;
            OUTPUT(STOCKS,rapscenarios)=INVEST.L
(STOCKS);
            OUTPUT("OBJ",rapscenarios)=OBJ.L;
            OUTPUT("MEAN",rapscenarios)
                 =SUM(STOCKS, MEAN(STOCKS) * INVEST.L
(STOCKS));
            OUTPUT("VAR",rapscenarios) = VAR;
            OUTPUT("STD",rapscenarios)=SQRT(VAR);
            OUTPUT("SHADPRICE",rapscenarios)
=INVESTAV.M;
            OUTPUT("IDLE",rapscenarios)=FUNDS-
INVESTAV.L
              );
 DISPLAY OUTPUT,solutionstatus;
```

Here we loop over the scenarios run (**RAPSCENARIOS)** and during that loop we load the GUSS saved investment levels into the original model investment variables using the statement

```
INVEST.L(STOCKS)=STOCKOUTPUT(RAPSCENARIOS,STOCKS);
```

along with the saved scenario dependent values of the objective function and the funds shadow prices.

```
OBJ.L=OBJLEVEL(RAPSCENARIOS);
INVESTAV.m = INVESTAVshadow(RAPSCENARIOS);
```

Finally in the loop a number of calculations are done placing results into a parameter named OUTPUT and after the loop the result is displayed as is the array holding the solution and model termination status.

Much more complex setups could be run.

Notes

- GUSS is not a solver and is not activated using normal solver choice methods such as option LP=GUSS or any such variants. Rather one uses a modification to the solve statement as discussed above.
- GUSS will cause the problems to be solved with the solver that is currently active in the GAMS instance. This may be specified using multiple ways as discussed here employing for example

© 2022 Prof. Bruce McCarl

OPTION NLP=CONOPT or LP=CPLEX or the like as well as through choice in the IDE or on the computer.

- GUSS has a number of additional options as discussed in http://www.gams.com/modlib/adddocs/gusspaper.pdf and in the GUSS section of the solver manual.

3.4.3.3.21 IPOPT/IPOPTH

IPOPT: Ipopt (Interior Point Optimizer) is an open-source solver for large-scale nonlinear programming. The code has been written primarily by Andreas W"achter, who is the COIN-OR project leader for Ipopt. GAMS/Ipopt uses MUMPS (http://graal.ens-lyon.fr/MUMPS) as linear solver.

For more information:
-the Ipopt web site https://projects.coin-or.org/Ipopt and
-the implementation paper A. W"achter and L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, Mathematical Programming 106(1), pp. 25-57, 2006.

A link to the Coin Ipopt (**I**nterior **P**oint **OPT**imizer, pronounced I-P-Opt) solver for large-scale nonlinear optimization. COINIPOPT is designed to find (local) solutions of mathematical optimization problems of the form:

```
min                          f(x)
x in R^n
s.t.          g_L      <=        g(x)  <=   g_U
              x_L  <=  x          <=   x_U
```

where

f(x)	is a real nonlinear objective function,
g(x)	is a set of real nonlinear constraint functions.
g_L	is the vector of lower bounds on the g(x) constraints
g_U	is the vector of upper bounds on the g(x) constraints
x_L	is the vector of lower bounds on the x variables
x_U	is the vector of upper bounds on the x variables

An in core link exists and has the solver name IPOPTD.

IPOPT was previously named COINIPOPT.

IPOPTH is a commercially licensed version that uses higher performance (but not Open Source) linear algebra routines (HSL).

3.4.3.3.22 JAMS

JAMS is a rename of the former solver EMP and handles EMP (Extended Mathematical Programming) problems.

© 2022 Prof. Bruce McCarl

JAMS reformulates
Bilevel Programs
Disjunctive Programs
Extended Nonlinear Programs
Embedded Complementarity Systems
Variational Inequalities

into established mathematical programming classes allowing access to existing solvers.

More details on JAMS and the EMP concept and function can be found in http://pages.cs.wisc.edu/~ferris/talks/focapo08.pdf and in JAMS.

JAMS comes free of charge with any licensed GAMS system but needs a subsolver to solve the generated models.

3.4.3.3.23 KESTREL

TKestrel is not really a solver but rather a link to external solvers via the NEOS Server from within your GAMS modeling environment. It also returns a solution that can be processed as with any local solver.

Starting with distribution 23.7 Kestrel is part of the GAMS distribution.

The solve statement using the GAMS/KESTREL solver invokes a client program that sends your problem to a solver running on one of the NEOS Server's remote computers. The results from the NEOS Server are eventually returned through Kestrel to GAMS, where you can view and manipulate them locally in the usual way.

Further information about KESTREL can be found in the solver manual or at the Kestrel page on the NEOS Server http://neos.mcs.anl.gov/neos/kestrel.html.

3.4.3.3.24 KNITRO

KNITRO is a solver for finding local solutions of MINLPs and continuous, smooth nonlinear optimization problems, with or without constraints from Ziena Optimization, Inc. KNITRO implements both interior (or barrier) and active-set type algorithms, and uses trust regions to promote convergence. KNITRO was designed for large problems with dimensions running into the hundreds of thousands. It converges quickly to minimize the number of objective function evaluations, and utilizes second derivative (Hessian) information to improve performance and robustness.

In terms of its MINLP capability: binary and integer variables are supported. Two algorithms are available, a non-linear branch and bound method and an implementation of the hybrid Quesada-Grossman method for convex MINLP. The Knitro MINLP code is designed for convex mixed integer programming and is a heuristic for nonconvex problems.

The manual is on **knitro**.

3.4.3.3.25 LGO

LGO - abbreviated from the name Lipschitz Global Optimizer - assists in the formulation and solution of the broad class of Global Optimization problems. LGO was developed by J. Pintér of Pintér Consulting Services, Inc. LGO integrates a suite of global and local scope solvers. These include (a) global adaptive partition and search (branch-and-bound); (b) adaptive global random search; (c) local (convex)

© 2022 Prof. Bruce McCarl

unconstrained optimization; and (d) local (convex) constrained optimization. LGO can solve LP, NLP, DNLP, RMINLP, and RMIP model types. The LGO manual is on **LGO**.

An in core link exists and has the solver name LGOD

3.4.3.3.26 LINDO/LINDOGLOBAL

LINDOGLOBAL: GAMS/LINDOGlobal finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDOGlobal supports most mathematical functions, including functions that are nonsmooth, such as abs(x) and or even discontinuous, such as floor(x).

The LINDO global optimization procedure (GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

The documentation for LINDOGLOBAL is at **LINDOGLOBAL**

3.4.3.3.27 LOGMIP

LogMIP solves linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for models involving discrete choices. LogMIP was been developed by A. Vecchietti, J.J. Gil and L. Catania at INGAR (Santa Fe-Argentina) and
Ignacio E. Grossmann at Carnegie Mellon University (Pittsburgh-USA). LogMIP comes free of charge with any licensed GAMS system but needs a subsolver to solve the generated MIP/MINLP models.

For more information see the solver manuals or http://www.logmip.ceride.gov.ar/eng/about/about.htm.

3.4.3.3.28 MILES

MILES is a solver for mixed complementarity problems and nonlinear systems of equations developed by T. Rutherford, University of Colorado. MILES is freely distributed with GAMS but is of lesser capability than the other available solvers. The solution procedure is a generalized Newton method with a backtracking line search. The MILES manual is on **MILES**.

There are two alternative named MILES versions.

MILESE
MILESOLD

© 2022 Prof. Bruce McCarl

3.4.3.3.28.1 MILESE

The newest version of MILES and the solver used when the word MILES is used.

3.4.3.3.28.2 MILESOLD

An older now discontinued version of MILES using an older interface.

3.4.3.3.29 MINOS

MINOS is a solver for large-scale nonlinear optimization (NLP) developed by B. Murtaugh and M. Saunders at Macquarie University and Stanford University. MINOS solves such problems using a reduced-gradient algorithm combined with a quasi-Newton algorithm. When constraints are nonlinear, MINOS employs a projected Lagrangian algorithm. This involves a sequence of major iterations, each of which requires the solution of a linearly constrained subproblem. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds. MINOS can solve LP, RMIP, NLP, DNLP, and RMINLP model types. The solver manual is **MINOS**. MINOSD is an in core version.

One alternative version is available.

3.4.3.3.29.1 MINOS5

An older version of MINOS.

3.4.3.3.30 MOSEK

MOSEK is a solver for linear, mixed-integer linear, and convex nonlinear mathematical optimization problems developed by Erling D. Andersen and Knud D. Andersen of MOSEK ApS as described at *http://www.mosek.com/*. MOSEK contains several optimization approaches designed to solve large-scale sparse problems. The current optimizers include:

- Interior-point optimizer for all continuous problems
- Conic interior-point optimizer for conic quadratic problems
- Simplex optimizer for linear problems
- Mixed-integer optimizer based on a branch and cut technology

MOSEK can solve LP, MIP, QCP, RQCP, MIQCP, RMIP, NLP, DNLP, and RMINLP model types. The NLP components it works on ordinarily should be highly convex as it can get stuck on non convex problems. The solver manual is **Mosek**.

A bare bones, free version of MOSEK is available and is called COINMOSEK.

.

3.4.3.3.31 MPSGE

MPSGE is not actually a solver but rather is a preprocessor that aids in the formulation and solution of

© 2022 Prof. Bruce McCarl

general equilibrium problems and operates as a subsystem to GAMS developed by T. Rutherford at University of Colorado. MPSGE is a library of function and Jacobian evaluation routines that generate an MCP problem that must be solved with a MCP solver. MPSGE, while in GAMS also involves a model definition language with alternative syntax and conventions that are unlike that used in the rest of GAMS. The modeling syntax and conventions are explained in **MPSGE** and additional material can be found at *http://www.mpsge.org/mainpage/mpsge*

3.4.3.3.32 MPS2GMS

GAMS distributes a software tool called MPS2GMS that will convert MPS files to GAMS. This software is found in the GAMS system directory and when run without arguments will deliver a small write-up.

3.4.3.3.33 MSNLP

MSNLP (Multi-Start NLP) is a stochastic search algorithm from Optimal Methods, Inc for global optimization problems. Like OQNLP, MSNLP uses a point generator to create candidate starting points for a local NLP solver. Algorithm performance depends strongly on the starting point generator. MSNLP implements a generator creating uniformly distributed points and the Smart Random Generator. This generator uses an initial coarse search to define a promising region within which random starting points are concentrated. Two variants of Smart Random are currently implemented, one using univariate normal distributions, the other using triangular distributions. MSNLP also comes the NLP solver LSGRG and is available as part of the Global Packages. Documentation in found in the **MSNLP Manual**.

3.4.3.3.34 NLPEC

NLPEC is a solver developed NLPEC is a solver developed jointly by GAMS Corporation and M. Ferris at UW-Madison that solves MPEC models. NLPEC works by reformulating the MPEC model as an NLP, solving the NLP using one of the GAMS NLP solvers, and then extracting the MPEC solution from the NLP solution. All of this happens automatically, although it is possible to access the intermediate NLP model. The reformulated models NLPEC produces are in scalar form. Many different reformulations (currently around 20) are supported by the NLPEC solver. MPEC models are notorious for their difficultly, but the combination of different reformulations and NLP solvers give users a good chance to solve them. The user guide is **nlpec**.

3.4.3.3.35 OSICplex

A bare bones, free version of CPLEX that comes free of charge with the GAMS Base system. General GAMS options (reslim, optcr, nodlim, iterlim) are supported. In addition an option file in the format required by the solver can be provided.

OSICPLEX was previously named COINCPLEX.

3.4.3.3.36 OSIGurobi

A bare bones, free version of GUROBI that comes free of charge with the GAMS Base system. General GAMS options (reslim, optcr, nodlim, iterlim) are supported. In addition an option file in the format required by the solver can be provided.

OSIGUROBI was previously known as COINGUROBI.

© 2022 Prof. Bruce McCarl

3.4.3.3.37 OSIMosek

A bare bones, free version of MOSEK that comes free of charge with the GAMS Base system. General GAMS options (reslim, optcr, nodlim, iterlim) are supported. In addition an option file in the format required by the solver can be provided.

OSIMOSEK was previously named COINMOSEK.

3.4.3.3.38 OSIXPRESS

A bare bones, free version of XPRESS that comes free of charge with the GAMS Base system. General GAMS options (reslim, optcr, nodlim, iterlim) are supported. In addition an option file in the format required by the solver can be provided.

OSIXPRESS was previously named COINXPRESS.

3.4.3.3.39 OSISOPLEX

OSISoPlex also called SOPLEX is a *Linear Programming (LP) solver* based on the revised simplex algorithm. It features preprocessing techniques, exploits sparsity, and offers primal and dual solving routines. It can be used as a standalone solver reading MPS or LP format files as well as embedded into other programs via a C++ class library.

OSISoPlex has been implemented as a part of Roland Wunderling's Ph.D. thesis *Paralleler und Objektorientierter Simplex-Algorithmus* (in German) and is available in source code. OSISoPlex is free for academic research and can be licensed for commercial use.

OSISOPLEX is now called SOPLEX in the solver capabilities matrix although both OSISOPLEX and SOPLEX work as named solvers.

3.4.3.3.40 PATH/PATHNLP

PATH is a MCP, CNS and NLP solver developed by M. Ferris, S. Dirkse and T. Munson at the University of Wisconsin. For MCP and CNS problems, PATH uses a generalization of Newton's method plus a linearization solved using a code related to Lemke's method. PATH can be used to solve MCP and CNS model types. The solver manual on PATH is found on **PATH** and additional details can be found on *http://www.neos-guide.org/content/complementarity-problems*.

There are 2 variants on PATH.

PATHC
PATHNLP

© 2022 Prof. Bruce McCarl

3.4.3.3.40.1 PATHC

PATHC is the latest version of PATH and is the one used when the solver named PATH is invoked.

3.4.3.3.40.2 PATHNLP

PATHNLP is a variant of PATH that can solve LP and NLP model types. Essentially, it automatically reformulates an NLP or LP problem as a complementarity problem and solves this using PATH. PATH then uses second order information in the solution of the model, which can result in greater solution efficiency. In addition, the marginal values are sometimes more exact than those provided by first-order methods. This approach can work better than other solvers on large, sparse models with many nonlinear variables and degrees of freedom. In these cases, the superbasics limit of other NLP codes can limit their effectiveness. PATHNLP solver allows the solution of certain previously unsolvable models (e.g. maximum entropy models). PATHNLP can be used with LP, RMIP, NLP, and RMINLP model types. The PATHNLP solver manual is essentially the PATH manual **PATH** but additional undocumented options exist. The solver manual is **PATHNLP** .

3.4.3.3.40.3 PATHOLD

PATHOLD is an older, now discontinued, version of PATH.

3.4.3.3.41 SBB

SBB is a solver for mixed integer nonlinear programming models developed by M. Bussieck of GAMS and A. Drud of ARKI Consulting & Development. It combines the branch and bound method known from mixed integer linear programming and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use

CONOPT
CONOPT2
MINOS
PATHNLP
SNOPT

as solvers for sub-models. SBB supports binary, integer, semicont, semiint, SOS1 and SOS2 variables. SBB employs a branch and bound algorithmic approach. SBB will solve MINLP model types.

3.4.3.3.42 SCENRED

SCENRED is a tool for the reduction of scenarios in a stochastic programs setting. SCENRED reduces the random scenario set determining a scenario subset of prescribed cardinality or accuracy and assigns optimal probabilities to the preserved scenarios. The reduced problem is then solved by a deterministic optimization algorithm using the GAMS solvers. The solver manual is on **scenred**.

3.4.3.3.43 SCIP

SCIP is a MIP Solver for the Constrained Integer Programming framework.

The code was developed at the Konrad-Zuse-Zentrum fllur Informationstechnik Berlin (ZIB) and was written primarily by T. Achterberg as part of the COIN project. It is distributed under the ZIB Academic License and is available free for academic users. The solver is discussed at the link SCIP

© 2022 Prof. Bruce McCarl

and in the solver reference manuals.

SCIP was previously known as COINSCIP.

3.4.3.3.44 SNOPT

SNOPT is a NLP solver developed by P. Gill University of California, San Diego along with W. Murray, and M. Saunders at Stanford University. SNOPT is suitable for large nonlinearly constrained problems with a modest number of degrees of freedom. SNOPT implements a sequential programming algorithm that uses a smooth augmented Lagrangian merit function and makes explicit provision for infeasibility in the original problem and in the quadratic programming sub-problems. SNOPT can solve LP, RMIP, NLP, DNLP, and RMINLP model types. The solver manual is on **SNOPT**.

3.4.3.3.45 SOPLEX

SoPlex is an optimization package for solving linear programming problems (LPs) based on the primal and dual revised simplex algorithms.

More on it appears at https://soplex.zib.de/.

It has been available in GAMS for sometime under the name OSISOPLEX and currently is covered in the solver guide only under that name.

3.4.3.3.46 XA

XA is a optimizer for solving linear, and mixed-integer problems developed by Sunset Software Technology *http://www.sunsetsoft.com/*. XA contains a modified primal simplex algorithm, a dual simplex algorithm, and an interior point barrier algorithm. For problems with integer variables, XA uses a branch and bound algorithm (with cuts) and supports specially ordered set variables SOS1, SOS2 as well as semi-continuous and semi-integer variables. XA solves LP, MIP and RMIP model types. The solver manual for XA is on **XA**.

An additional variant of XA exists.

3.4.3.3.46.1 XAPAR

XAPAR is a parallel processor version of XA.

3.4.3.3.47 XPRESS

XPRESS is the XPRESS-MP Optimization Subroutine Library developed by *Dash Optimization* and integrates a simplex-based LP solver, a MIP module, a barrier module implementing an interior point algorithm for LP problems and a quadratic algorithm. XPRESS solves LP, MIP, QCP, RQCP, MIQCP and RMIP model types. QCP problems are limited to those with quadratic objectives. The solver manual for XPRESS is on **XPRESS**.

A free bare bones versions is available as COINXPRESS.

© 2022 Prof. Bruce McCarl

3.4.3.4 Choosing a solver

GAMS offers a number of choices to solve a model and the user may switch solvers at their discretion providing they have appropriate licenses. Mechanically, there are several ways to switch as discussed in the Variables, Equations, Models and Solves chapter.

3.5 Standard Output

The standard output from GAMS contains many components. Here I discuss standard and optional components of the output file as well as error messages. I will also cover procedures for controlling the amount of output produced.

This chapter does not cover the output that can be generated by the user through Display or Put statements. Those topics are covered in the Improving Output via Report Writing and in Output via Put Commands chapters.

Where is my output? LOG and LST files
GAMS phases and output generated
Compilation phase output
Execution output
Output produced by a solve statement
Managing output pages
Managing output volume
Adding slack variables to the output
Sending messages to the LOG file

3.5.1 Where is my output? LOG and LST files

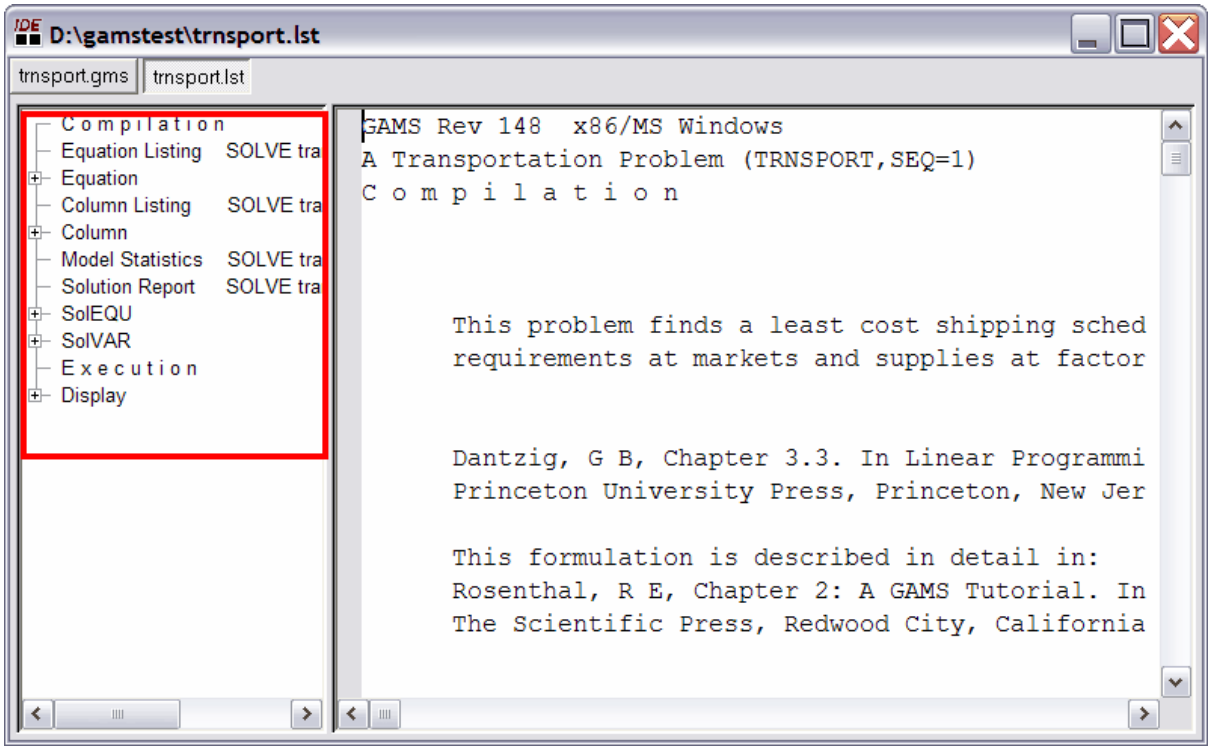
When GAMS runs an output file is automatically generated. That output file has the file extension LST. The first part of the LST file name is the first part of the name of the GAMS input file. Thus, if the file myfile.gms is the file run through GAMS the output will be on myfile.LST. This output file is an ASCII text file in the courier font that can be loaded into a text editor. GAMS also generates a LXI file (in this case myfile.LXI that contains navigation information that is used by the IDE in moving generating the navigation window that permits around the LST file.

Output also appears on the LOG file particularly if one is using the IDE. That output summarizes the run results and also can serve as a navigation aid when using the IDE. The LOG file does contain information relevant to the solution and error status. It is named using the same practice as with the LST file but the file extension is LOG i.e. myfile.LOG. It also is reproduced in the process window of the IDE.

3.5.2 Output overview and navigation

When GAMS runs it automatically opens the LST file and a navigation window as identified in the red box below. By clicking on lines in the lst file navigation window you can access program output both in general and at particular locations.

© 2022 Prof. Bruce McCarl



The positioning of the cursor in the LST file is determined by the type of line you click on. A list of types of lines typically in the LST file is given below and the contents of this will be defined below..

© 2022 Prof. Bruce McCarl

Name of Line in LST File Navigation Window	Function and Destination When Clicked.
Compilation	Jumps to top of echo print in LST file
Error Messages	Jumps to list of error messages when compilation messages incurred
Equation listing	Jumps to list of equation contents as illustrated.in the output section
Equation	Expandable allowing jump to beginning of list of contents for each individual equation block with contents as shown.in the output section
Variable listing	Jumps to list of variable contents as illustrated.in the output section
Variable	Expandable allowing jump to beginning of list of contents for each individual equation block with contents as shown.in the output section
Model statistics	Jumps to model statistics part of LST file as illustrated.in the output section
Solution Report	Jumps to model summary solution report
SolEQU	Expandable allowing jump to beginning of list of solution for each individual equation block with contents as shown.in the output section
SolVAR	Expandable allowing jump to beginning of list of solution for each individual variable block with contents as shown.in the output section
Execution	Jumps to beginning of post solve execution
Display	Expandable allowing jump to displays of specific parameters and other items

The width of the LST file navigation window is controlled by the user and can be narrowed as it has been above.

The data for this is stored in a LXI file that will be resident in your project directory with the same root as the associated LST file.

© 2022 Prof. Bruce McCarl

3.5.3 GAMS phases and output generated

When asked to run, GAMS passes through a program file several times. The main passes it makes are the

- Compilation phase where the LST file is written containing an echo print of the source file possibly containing error messages, along with lists of GAMS objects, and cross reference maps.
- Execution phase where output containing displays and execution error messages is added to the LST file.
- Generation phase where output containing listings of equations and variables along with generation execution error messages is added to the LST file.
- Solution phase where an external solver program deals with the model and creates output relative to the solution process is added to the LST file.
- Post solution phase where output on the model solution and other user created displays is added to the LST file.

3.5.4 Compilation phase output

During the compilation phase GAMS lists back the original program including line numbers, incorporates a marking of and explanation of any errors detected, creates object lists and a cross reference map.

Echo print of the input file
Symbol reference map
Symbol listing
Unique element list
Unique element cross reference

3.5.4.1 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 lines numbers added. An echo print of the file **shortmodel.gms** follows:

```
 2  *Example of a comment
    Comments in $On/off text are not given line numbers
 6  SET PROCESS   PRODUCTION PROCESSES /makechair,maketable,makelamp/
 7      RESOURCE  TYPES OF RESOURCES   /plantcap,salecontrct/;
 8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
 9                /makechair 6.5 ,maketable 3, makelamp 0.5/
10            Yield(process)  yields per unit of the process
11                /Makechair 2   ,maketable 6 ,makelamp 3/
12          PRODCOST(PROCESS)     COST BY PROCESS
13                /Makechair 10  ,Maketable 6, Makelamp 1/
14          RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
15               /plantcap 10 ,salecontrct 3/;
16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
17                      Makechair   Maketable  Makelamp
```

© 2022 Prof. Bruce McCarl

```
18       plantcap         3          2          1.1
19       salecontrct      1         -1;
20  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
21  VARIABLES          PROFIT              TOTALPROFIT;
22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE ;
24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
25                         -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
27                    *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
28  MODEL RESALLOC /ALL/;
29  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
34  display solprod;
```

Notes:

- The **line number** refers to the input file physical line number of each statement starting with line 1. However, this is not true if other files are incorporated using the commands for file inclusion or saves and restarts. In those cases the line numbers are sequential starting from 1 in the first file with line numbers added at points of inclusion for the full file.
- Some lines are left out of the LST file echo print. In particular, any $directives inserted in the source file are not listed even though the line number count is advanced for their presence. These will only be listed if a directive to list them is enabled ($Ondollar), or if they contain errors. Thus, since in the **shortmodel.gms** example the source file has a $ command in line 1 then the above listing begins with line **2**. $Offdollar stops echo print of dollar command options in LST file.
- GAMS does not echo back line numbers for entries enclosed in $Ontext and $Offtext. Thus, since the **shortmodel.gms** example source file contains such commands in lines 3 and 5 with comments inserted between (line 4) the LST file contains the comment in the echo print output but the line numbers are suppressed and the dollar commands skipped resulting in the line numbering skipping from line **2** to line **6**.
- The $Offlisting directive will turn off the echo print of any lines appearing after it in the input file and $Onlisting will turn the echo print back on. Thus, since **shortmodel.gms** has such commands in line 30 and 33, the line numbers skip from line **29** to line **34** and the lines in between are not shown.
- Lines in the echo print can be caused to be double spaced using $double and then reset to single spacing using $single.
- Include files are ordinarily copied into the echo print but can be suppressed using $Offinclude. They can be subsequently reactivated using $Oninclude.

3.5.4.1.1 Compilation phase error messages

During the compilation phase GAMS carefully checks the input file for consistency with GAMS syntax and semantics. In turn, GAMS provides feedback and suggestions about how to correct errors or avoid ambiguities as discussed in the Compiler Errors chapter.

Several hundred different types of errors can be detected during compilation. 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 as shown in the

© 2022 Prof. Bruce McCarl

Compiler Errors chapter.

When a compilation error is discovered a line is inserted in the echo print marked with four asterisks '****'. Also in that line a $-symbol is inserted followed by a number which cross references to a list of error conditions followed by a brief explanation of the nature of the error. This $ is inserted immediately below the place in the line where the error arose (usually to the right). If more than one error is encountered on a line the $-signs may be suppressed and error numbers squeezed together. GAMS will not list more than 10 errors on any one line. The IDE also provides help in locating errors as discussed in the GAMS Usage chapter.

When errors are present, the LST file contains a list of all the different types of errors encountered by error number just after the end of the echo print listing. That list which includes a description of the probable cause of each error. The error messages are generally self-explanatory and will not be listed here. However, I do provide a list of the most common ones and their cause in the Compiler Errors chapter. These messages can be repositioned as discussed below.

Example:

The example **shorterr.gms** illustrates the general reporting format for compiler errors. The part of shorterr.LST relevant to errors is:

```
  6  SET PROCESS   PRODUCTION PROCESSES /makechair,maketable,makelamp/
  7      RESOURCE  TYPES OF RESOURCES   /plantcap,salecontrct/;
  8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
  9                /makechair 6.5 ,maketable 3, makelamp 0.5/
 10            Yield(process)  yields per unit of the process
 11                /Makechair 2   ,maketable 6 ,makelamp 3/
 12          PRODCOST(PROCESS)     COST BY PROCESS
 13                /Makechair 10  ,Maketable 6, Makelamp 1/
 14          RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
 15               /plantcap 10 ,salecontrct 3/;
 16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
 17                     Makechair   Maketable  Makelamp
 18       plantcap         3          2          1.1
 19       salecontrct      1         -1;
 20  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
 21  VARIABLES          PROFIT              TOTALPROFIT;
 22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
 23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE
 24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****      $96                          $2   $195      $96
 25                        -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
****                       $409
 26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
 27                    *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
 28  MODEL RESALLOC /ALL/;
 29  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                        $257
 32  solprod(PROCESS)= PRODUCTION.l(PROCESS);
****                               $141
```

© 2022 Prof. Bruce McCarl

```
  34  display solprod;

Error Messages
  2  Identifier expected
 96  Blank needed between identifier and text
       (-or- illegal character in identifier)
       (-or- check for missing ';' on previous line)
141  Symbol neither initialized nor assigned
        A wild shot: You may have spurious commas in the explanatory
        text of a declaration. Check symbol reference list.
195  Symbol redefined with a different type
257  Solve statement not checked because of previous errors
409  Unrecognizable item - skip to find a new statement
       looking for a ';' or a key word to get started again

**** 7 ERROR(S)   0 WARNING(S)
```

Notes:

- The lines containing the **** mark the errors encountered and also contain numbered error message references.
- The $ insertions cross reference the error messages to the listing of descriptions of the conditions at the end of the echo print while also indicating the placement of the error in the input file. In particular the lines

```
  23                    AVAILABLE(RESOURCE) RESOURCES AVAILABLE
  24  OBJT.. PROFIT=E=  SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****      $96                        $2   $195      $96
```

  show that error 96 has occurred in line 24

```
Blank needed between identifier and text
(-or- illegal character in identifier)
 (-or- check for missing ';' on previous line)
```

  at the position of the $ markers. The first error $96 occurs at the position of the .. in the line above and the associated message shows GAMS is expecting something different. In this case the problem arises because of the line before as the third line of the error message suggests. Namely the line before was not ended with a semicolon.

- The example shows one of the common findings when dealing with compilation errors. Namely one error typically proliferates throughout the code causing may other errors. In this case the only error in the code is the omission of a semicolon (;) at the end of line 23.

```
23 AVAILABLE(RESOURCE) RESOURCES;
```

  Typically many more compilation errors are marked than truly exist and can often be traced back to just one specific omission or error in the GAMS input.

- The almost certain proliferation of errors caused by early errors in the GAMS program means it is always advisable to check carefully from the top of the echo print finding and repairing the cause of

© 2022 Prof. Bruce McCarl

the first few errors in the code if one can figure them out then rerunning.

- The example also shows an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions with baffling explanations.
- One very common error is the omission of a semi colon.
- One can insert their own error messages using $abort or $error. The **** marker can be changed using $Stars.

3.5.4.1.1.1 Repositioning error messages

It is possible to reposition where the explanation of the errors appears. In particular, the explanation location can be altered so the error messages appear just below the place the error is found mixed in with the source listing. This is done by using the option errmsg=1 in the GAMS command line. This can be imposed in a couple of ways.

- One can call GAMS with the command line parameter **errmsg=1**

```
gams mymodel errmsg=1
```

When using the IDE this is placed in the GAMS command box in the upper right hand corner as discussed in the Gamside chapter or if wanted for all models in the file option choice under the execute tab in the box for additional GAMS parameters.

- One can alter the system level defaults as discussed in the Customizing GAMS chapter by entering this line in the file gmsprm95.txt on basic windows machines. That file is called gmsprmnt.txt on NT machines and gmsprmun.txt on Unix/Linux machines. The resultant file looks something like

```
*****************************************************
* GAMS 2.50 Default Parameterfile for Windows NT                *
* Gams Development Corp.                                        *
* Date : 20 Mar, 1998                                           *
*****************************************************
* entries required by CMEX, put in by gams.exe:
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
errmsg=1
ps=9999
optfile=1
```

In turn, the output looks like the following

```
   6  SET PROCES   PRODUCTION PROCESSES /makechair,maketable,makelamp/
   7      RESOURCE  TYPES OF RESOURCES   /plantcap,salecontrct/;
   8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
****                          $120
```

© 2022 Prof. Bruce McCarl

```
**** 120  Unknown identifier entered as set
   9               /makechair 6.5 ,maketable 3, makelamp 0.5/
  10          PRODCOST(PROCESS)     COST BY PROCESS
  11               /Makechair 10  ,Maketable 6, Makelamp 1/
****                        $361
**** 361  Values for domain 1 are unknown - no checking possible
  12          RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
  13               /plantcap 10 ,salecontrct 3/;
  14  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
****                                            $362
**** 362  Values for domain 2 are unknown - no checking possible
  15                     Makechair   Maketable  Makelamp
  16       plantcap         3          2          1.1
  17       salecontrct      1         -1;
  18  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
  19  VARIABLES          PROFIT              TOTALPROFIT;
  20  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
  21                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE;
  22  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****                                                $140
**** 140  Unknown symbol
```

3.5.4.2 Symbol reference map

The next part of the GAMS output is the symbol cross reference map. This lists the named items (Sets, Parameters, Variables, Equations, Models, Files, Acronyms) in alphabetical order, identifies them as to type, shows the line numbers where the symbols appear, and classifies each appearance.

The symbol reference map for the **shortmodel.gms** example is

```
SYMBOL      TYPE   REFERENCES
AVAILABLE   EQU    declared       23  defined       26 impl-asn       29
                        ref       28
OBJT        EQU    declared       22  defined       24 impl-asn       29
                        ref       28
ite         ACRNM  declared       36  defined       36
PRICE       PARAM  declared        8  defined        9      ref       24
PROCESS     SET    declared        6  defined        6      ref        8
                       10       12       16       20     2*24     2*25
                       26       27  control       24       26
PRODCOST    PARAM  declared       12  defined       13      ref       25
PRODUCTION  VAR    declared       20 impl-asn       29      ref       25
                       27
PROFIT      VAR    declared       21 impl-asn       29      ref       24
                       29
putfile     FILE   declared       37  defined       38
RESALLOC    MODEL  declared       28  defined       28 impl-asn       29
                        ref       29
RESORAVAIL  PARAM  declared       14  defined       15      ref       27
RESOURCE    SET    declared        7  defined        7      ref       14
```

© 2022 Prof. Bruce McCarl

```
                          16       23       26       27  control       26
RESOURUSE    PARAM  declared       16  defined       16      ref       26
solprod      PARAM       ref       34       38
Yield        PARAM  declared       10  defined       11      ref       24
```

The symbol column is an all inclusive alphabetical list of **all named items** in the GAMS program. The type column gives a categorical classification of the basic nature of the symbol using abbreviation codes as follows

TYPE	Type of GAMS item
ACRNM	Acronym
EQU	Equation
FILE	Put file
MODEL	Model
PARAM	Parameter (also Table, Scalar)
SET	Set
VAR	Variable

The symbol map also contains a line number indexed list of references to the items, grouped by type of reference which are abbreviated as below

© 2022 Prof. Bruce McCarl

Reference	Description
DECLARED	This identifies lines where the named item is declared in a Set, Parameter, Table, Scalar, Variable, Equation, Acronym, File or Model command. This will be the first appearance.
DEFINED	This identifies the line number where explicit data are entered (in a table or a data list between slashes) or in the case of an equation where the .. specification begins.
ASSIGNED	This identifies lines where this item has data put into it by being on the left hand side of an assignment statement.
IMPL-ASN	This identifies lines where an equation or variable has data put into it by the results of a solve statement.
CONTROL	This identifies lines where this set is used as the driving index either in an assignment, equation, loop or other indexed operation (`sum`, `prod`, `smin` or `smax`).
REF	This identifies lines where this item is used either on the right hand side of an assignment, in a `display`, in an `equation`, in a `model`, in a `put statement` or in a `solve statement`.

Notes:

- The symbol cross reference map does not always appear. In particular, it is suppressed by default in the IDE and, if desired, needs to be requested by using the command 6.
- In GAMS Dos and Unix the reference map always appears and if one wishes to suppress it one uses $Offsymxref.
- The reference map is often large and not always useful given the capabilities of today's text editors. It is usually suppressed in most models (with $Offsymxref).
- This map can be useful in model development, documentation preparation, and to make sure you do not have items that are present but never used (insuring all items have ref, impl-asn, or control entries present).

3.5.4.3 Symbol listing

The next output item is called the "symbol listing" and contains a classified list of all the named items along with their explanatory text. The classification is by the main object types (Sets, Parameters, Equations, Variables, and Models) is a useful way of obtaining information on item definitions provided explanatory names and explanatory text has been used as argued in the Writing Models and Good Modeling Practices chapter.

The symbol listing for the **shortmodel.gms** example is

SETS

© 2022 Prof. Bruce McCarl

```
   PROCESS     PRODUCTION PROCESSES
   RESOURCE    TYPES OF RESOURCES
ACRONYMS
   ite         test an acronym
PARAMETERS
   PRICE       PRODUCT PRICES BY PROCESS
   PRODCOST    COST BY PROCESS
   RESORAVAIL  RESOURCE AVAILABLITY
   RESOURUSE   RESOURCE USAGE
   solprod     report of production
   Yield       yields per unit of the process
VARIABLES
   PRODUCTION  ITEMS PRODUCED BY PROCESS
   PROFIT      TOTALPROFIT
EQUATIONS
   AVAILABLE   RESOURCES AVAILABLE
   OBJT        OBJECTIVE FUNCTION ( PROFIT )
MODELS
   RESALLOC
FILES
   putfile     test a file statement
```

Notes:

- The symbol list does not always appear. In particular, it is suppressed in the IDE and, if desired, needs to be requested by using the command $Onsymlist.
- In GAMS runs outside the IDE the symbol list always appears and if one wishes to suppress it one uses $Offsymlist.
- The symbol list is useful in model comprehension and documentation preparation.
- The capitalization in the symbol list is controlled by the rules explained in the Rules for Item Capitalization and Ordering chapter.
- Again long explanatory names and explanatory text really improve the utility of this list (see the Writing Models and Good Modeling Practices chapter for more on this point).

3.5.4.4 Unique element list

The next output item is the "Unique Element List" which contains a list of all set element known to the GAMS program in the order and capitalization style in which they will appear in the output. All unique elements are first listed in the entry order (as discussed in the Rules for Item Capitalization and Ordering chapter) and then in sorted order.

The lists for the **shortmodel.gms** example are:

```
Unique Elements in Entry Order
   1 makechair   maketable   makelamp    plantcap    salecontrct

Unique Elements in Sorted Order
```

© 2022 Prof. Bruce McCarl

```
1 makechair    makelamp    maketable    plantcap    salecontrct
```

Notes:

- The unique element list does not appear unless requested. If desired, it needs to be requested by using the command $Onuellist. $Offuellist removes the unique element listing from LST file.
- The capitalization and order of the UELs in the list and in all GAMS output is controlled by the rules explained in the Rules for Item Capitalization and Ordering chapter.
- The UEL list is probably only useful to check capitalization and ordering in an effort to improve output appearance and make sure nothing is omitted.

3.5.4.5 Unique element cross reference

The next output item is the unique element cross reference map that identifies the line numbers where all unique set elements in the GAMS program are declared and referenced.

```
ELEMENT       REFERENCES

makechair     declared      6      ref       9       11       13       17
makelamp      declared      6      ref       9       11       13       17
maketable     declared      6      ref       9       11       13       17
plantcap      declared      7      ref      15       18
salecontrct   declared      7      ref      15       19
```

Notes:

- The declared entry shows the first place the name appears.
- The ref entry shows where the set element is used.
- The UEL map is probably only useful to make sure nothing is omitted and to make sure you do not have items that are present but never used (insuring all items have ref entries present).
- The capitalization of the UEL's in the map and in all GAMS output is controlled by the rules explained in the Rules for Ordering and Capitalization chapter.
- The unique element cross reference map does not appear unless requested. If desired, it needs to be requested by using the command $Onuelxref. $Offuelxref removes unique element cross reference from LST file.

3.5.5 Execution output

GAMS provides output to the LST file while executing (performing data manipulations) from display statements and execution errors. Only brief discussion of displays are presented here. More coverage is given in the Report writing chapter.

Display output
Execution error output
Symptoms of the presence of an execution error

© 2022 Prof. Bruce McCarl

3.5.5.1 Display output

Users can employ display to create an entry in the LST file with the nonzero data for program items. The output from the display statement on line 34 for the **shortmodel.gms** example is shown below. The format of the display statement output can be altered as discussed in the Report writing chapter.

```
----     34 PARAMETER solprod  report of production
maketable 5.000
```

3.5.5.2 Execution error output

If errors are detected because of illegal data operations, GAMS will generate error messages. During GAMS usage one can encounter execution errors.

Generally these occur either during GAMS execution due to GAMS limits or math problems and are marked in the LST file with ****. The Fixing Execution Errors chapter shows how to find and fix these.

3.5.5.3 Symptoms of the presence of an execution error

When a job runs and execution errors are present the first indication is in the contents of the run summary sent to the screen, the LOG file and the IDE process window. For the example **executcl.gms** the LOG file appears as follows

```
--- Starting compilation
--- EXECUTCL.GMS(12) 1 Mb
--- Starting execution
--- EXECUTCL.GMS(11) 1 Mb 1 Error
*** Exec Error 10 at line 11
    Illegal arguments in ** operation
--- EXECUTCL.GMS(11) 1 Mb 2 Errors
*** Exec Error 0 at line 11
    Division by zero
--- EXECUTCL.GMS(12) 1 Mb 2 Errors
*** Status: Execution error(s)
```

showing the presence of two types of execution errors. The Fixing Execution Errors chapter shows how to find and fix these.

3.5.6 Output produced by a solve statement

When a Solve statement is executed another set of output is included in the LST file. This consists of a model generation error listing, equation listing, variable listing, model characteristics statistics output, model generation time report, solve summary report, solver report, and a variable and equation solution listing.

Model generation error listing
Equation listing
Variable listing
Model characteristics statistics

© 2022 Prof. Bruce McCarl

Model generation time
Solve summary
The variable and equation solution listing
Ranging analysis
Final execution summary
Report summary
File summary

3.5.6.1 Model generation error listing

The GAMS output next contains execution errors found during model generation. These are numerical calculation or model structure errors. The Fixing Execution Errors chapter shows how to find and fix these.

Numerical calculation errors involve improper exponentiation (such as raising a negative number to a real power), logs of negative numbers, or division by zero. Model structure errors involve equations improperly set up i.e. ones that are inherently infeasible or the wrong solver is being used.

Discovery of execution errors is sometimes very straight forward, but can, at other times, be fairly involved. Namely an error in the middle of a multi-dimensional equation block and/or in a multi-dimensional equation term within a block, can be difficult. The most practical way of finding such errors is to use the LIMROW/LIMCOL option commands.

Consider the example **executmd.gms**.

```
sets elems /s1*s25/
parameter data1(elems)   data to be exponentiated
          datadiv(elems) divisors
          datamult(elems)     x limits;
data1(elems)=1;
data1("s20")=-1;
datadiv(elems)=1;
datadiv("s21")=0;
datamult(elems)=1;
datamult("s22")=0;
positive variables x(elems) variables
variables  obj;
equations  objr        objective with bad exponentiation
           xlim(elems) constraints with bad divisor
 nonlin      nonlinear constrint in LP;
objr.. obj=e=sum(elems,data1(elems)**2.1*x(elems));
xlim(elems).. datamult(elems)/datadiv(elems)*x(elems)=e=1;
nonlin..    sum(elems,sqr(x(elems)))e=1;
model executerr /all/
option limrow=30; option limcol=30;
solve executerr using lp maximizing obj;
```

When this is run through GAMS I find execution errors where the LOG file contains

```
--- Generating model EXECUTERR
```

```
--- EXECUTMD.GMS(16) 134 Kb
*** ExecError 10 at Line 16
    ILLEGAL ARGUMENTS IN ** OPERATION
--- EXECUTMD.GMS(17) 134 Kb 1 Errors
*** ExecError 0 at Line 17
    DIVISION BY ZERO
*** ExecError 28 at Line 17
    EQUATION INFEASIBLE DUE TO RHS VALUE
--- EXECUTMD.GMS(20) 134 Kb 3 Errors
*** SOLVE aborted
*** Status: Execution error(s)
```

and the LST file

```
**** EXECUTION ERROR 10 AT LINE 16 .. ILLEGAL ARGUMENTS IN ** OPERATION

---- OBJR        =E=  objective with bad exponentiation

OBJR..  - X(s1) - X(s2) - X(s3) - X(s4) - X(s5) - X(s6) - X(s7) - X(s8)
      - X(s9) - X(s10) - X(s11) - X(s12) - X(s13) - X(s14) - X(s15) - X(s16)
      - X(s17) - X(s18) - X(s19) + UNDF*X(s20) - X(s21) - X(s22) - X(s23)
      - X(s24) - X(s25) + OBJ =E= UNDF ; (LHS = UNDF, INFES = UNDF ***)

**** EXECUTION ERROR 0 AT LINE 17 .. DIVISION BY ZERO

**** EXECUTION ERROR 28 AT LINE 17 .. EQUATION INFEASIBLE DUE TO RHS VALUE

**** INFEASIBLE EQUATIONS ...

---- XLIM        =E=  constraints with bad divisor

XLIM(s22)..  0 =E= 1 ; (LHS = 0, INFES = 1 ***)
```

In this example there are execution errors:

- in the objective function for the "S20" element where the code is exponentiating a negative constant to a real power (because of the assignment in line 9);
- in the XLIM "S22" constraint where the code is setting zero equal to one which results in an infeasible constraint (because of the assignment in line 15).
- in the XLIM constraint associated with element "S21" where the code is dividing by zero (because of the assignment in line 11); but this example does not at first display the divide by zero error. To get it you have to fix the infeasibility and run again. Then you get

```
XLIM(s21)..  UNDF*X(s21) =E= UNDF ; (LHS = UNDF, INFES = UNDF ***)
```

3.5.6.2 Equation listing

The next output item is the equation listing, which is marked with that subtitle on the output file and is

© 2022 Prof. Bruce McCarl

controlled by Option Limrow. 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 a device for studying this extremely important question (as are many of the features of **GAMSCHK**). This component of the LST file shows specific equations generated within the model when the current values of the sets and parameters are plugged into the general algebraic form of the model. For the **shortmodel.gms** example it is

```
---- OBJT  =E=  OBJECTIVE FUNCTION ( PROFIT )

OBJT..  - 3*PRODUCTION(makechair) - 12*PRODUCTION(maketable)
      - 0.5*PRODUCTION(makelamp) + PROFIT =E= 0 ; (LHS = 0)

---- AVAILABLE  =L=  RESOURCES AVAILABLE

AVAILABLE(plantcap)..  3*PRODUCTION(makechair) + 2*PRODUCTION(maketable)
      + 1.1*PRODUCTION(makelamp) =L= 10 ; (LHS = 0)

AVAILABLE(salecontrct)..  PRODUCTION(makechair) - PRODUCTION(maketable) =L= 3
      ; (LHS = 0)
```

Notes:

- The equation listing shows the cases of each constraint, variables that appear, numerical values of the individual coefficients and the equation type.
- The constant term on the right-hand-side value collapses all constants in the equation with appropriate sign alterations.
- The equation listing for each equation block is set off with four dashes ---- to facilitate mechanical searching.
- The coefficients are shown with four decimal places if needed, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers being displayed as zero and to allow large numbers.
- The default output is a maximum of three index cases for each equation block. To change the default, insert an option statement prior to the solve statement: option limrow = r ; where r is the desired number of equation cases in a block to be displayed. Equations that are infeasible at the starting point are marked with three asterisks (***) as in the NLP listing just below.
- The order in which the equations and equation cases are listed depends on the form of the **model** statement. If /**all**/ was used then the order is determined by the entry order of the equations and the placement of set elements in the unique element list as discussed in the Rules for Item Capitalization and Ordering chapter. On the other hand if the equations are explicitly listed in the **model** statement, then the listing will be in the order in that list with elements listed according to the unique element list.
- Nonlinear terms are also included here but are marked and are local evaluations. Namely, nonlinear coefficients are enclosed in parentheses, and the value of the coefficient depends on the level attributes (.l values) of the variables. The listing shows the partial derivative of each variable evaluated at their current level values and there is an implicit unlisted constant involved with the function evaluations. For the example **simpnlp.gms** with the equation

© 2022 Prof. Bruce McCarl

```
Eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2;
```

At the starting point

```
x.l = 2; y.l = 3 ;
```

The equation listing contains

```
Eq1.. (221)*x + (216.1667)*y =2= ; (lhs = 225.5 ***)
```

For further discussion see the NLP and MCP chapter.

3.5.6.3 Variable listing

The next output section is the variable listing and is controlled by option Limcol. This lists the individual coefficients for each variable. For the **shortmodel.gms** example it is

```
---- PRODUCTION  ITEMS PRODUCED BY PROCESS

PRODUCTION(makechair)
                 (.LO, .L, .UP = 0, 0, +INF)
       -3        OBJT
        3        AVAILABLE(plantcap)
        1        AVAILABLE(salecontrct)
PRODUCTION(maketable)
                 (.LO, .L, .UP = 0, 0, +INF)
      -12        OBJT
        2        AVAILABLE(plantcap)
       -1        AVAILABLE(salecontrct)
PRODUCTION(makelamp)
                 (.LO, .L, .UP = 0, 0, +INF)
       -0.5      OBJT
        1.1      AVAILABLE(plantcap)

---- PROFIT  TOTALPROFIT

PROFIT
                 (.LO, .L, .UP = -INF, 0, +INF)
        1        OBJT
```

Notes:

- The variable listing shows each case of each variable including the equations in which appears, and the numerical values of the individual coefficients.
- The listing also shows the lower bound (.lo), upper bound (.up) and current level (.l) values for each variable.
- The listing for each variable block is set off with four dashes ---- to facilitate mechanical searching.
- The numerical entries are formatted with up to four decimal places if needed, but trailing zeroes

© 2022 Prof. Bruce McCarl

following the decimal point are suppressed. E-format is used to prevent small numbers being displayed as zero and to allow display of large numbers.

- The default output is a maximum of three set index cases for each generic variable. To change the default, insert an option statement prior to the solve statement: option limcol = r ; where r is the desired number of variable cases to be displayed under each variable block.
- The order in which the variable cases and the variable blocks are listed is determined by the entry order of the variables and the placement of set elements in the unique element list as discussed in the Rules for Item Capitalization and Ordering chapter.
- Nonlinear terms are included here but are marked. Namely, nonlinear coefficients are enclosed in parentheses, and the value of the coefficient depends on the activity levels (.l values) of the variables. The listing shows the partial derivative of each variable evaluated at their current level value attribute and there is an implicit unlisted constant involved with the function evaluations. For the example **simpnlp.gms** with the equation

```
Eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2;
```

At the starting point

```
x.l = 2; y.l = 3 ;
```

The variable listing contains

```
---- x
x
                (.LO, .L, .UP = -INF, 2, 10)
     (221)      eq1

---- y
y
                (.LO, .L, .UP = -INF, 3, 10)
     (216.1667)      eq1
```

3.5.6.4 Model characteristics statistics

The next item of output is the model size statistics. For the **shortmodel.gms** example it is

```
MODEL STATISTICS

BLOCKS OF EQUATIONS       2     SINGLE EQUATIONS        3
BLOCKS OF VARIABLES       2     SINGLE VARIABLES        4
NONZERO ELEMENTS          9
```

While for the example **simpnlp.gms** it is

```
MODEL STATISTICS

BLOCKS OF EQUATIONS       1     SINGLE EQUATIONS        1
```

© 2022 Prof. Bruce McCarl

```
BLOCKS OF VARIABLES       2     SINGLE VARIABLES        2
NONZERO ELEMENTS          2     NON LINEAR N-Z          2
DERIVATIVE POOL           6     CONSTANT POOL          11
CODE LENGTH              54
```

This output provides details on the size and nonlinearly of the model.

The block counts refer to GAMS equations and variables, the single counts to individual rows and columns in the problem generated. The NONZERO ELEMENTS entry refers to the number of nonzero coefficients in the problem matrix.

There are several entries that provide additional information about nonlinear models.

- The NON LINEAR N-Z entry refers to the number of nonlinear coefficient entries in the model.
- The CODE LENGTH entry reports on the complexity of the nonlinearly of the model and is really telling how much code GAMS passes to the nonlinear solver which describes all the nonlinear terms in the model.
- The DERIVATIVE POOL and CONSTANT POOL provide more information about the nonlinear information passed to the nonlinear solver.

3.5.6.5 Model generation time

GAMS reports CPU or clock time usage and memory usage are next as follows

```
GENERATION TIME      =        0.010 SECONDS    1.4 Mb      WIN200-121
```

The generation time is the time used since compilation finished. This includes the time spent in generating the model. The measurement units are ordinary clock time on personal computers, and central processor usage (cpu) time on other machines. Memory use is given in megabytes.

3.5.6.6 Solve summary

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.

3.5.6.6.1 Common solver report

The section of the solve summary that is common for all solvers is shown below.

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

     MODEL   RESALLOC            OBJECTIVE  PROFIT
     TYPE    LP                  DIRECTION  MAXIMIZE
     SOLVER  BDMLP               FROM LINE  29

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE               60.0000

 RESOURCE USAGE, LIMIT          0.070      1000.000
```

© 2022 Prof. Bruce McCarl

```
ITERATION COUNT, LIMIT         2         10000
```

This can be found mechanically by searching for S O L V E. This has a number of parts

- Name of the model being solved,
- Name of the variable optimized (objective),
- Problem type
- Direction of optimization
- **Solver name**
- **Line number** where the solve statement appears.
- **A**n indication of the solver and model status. As discussed in the Model Attributes chapter.
- **The** objective value.
- Resource usage - the amount of cpu time (in seconds) taken by the solver, as well as the time limit allowed for the solver.
- The iteration count, and the iteration upper limit.
- In a non linear model one also gets counts of evaluation errors providing the number of numerical errors encountered by the solver, as well as the upper limit allowed. These errors result due to numerical problems like division by 0.

3.5.6.6.2 Solver report

The next section in the LST 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: BDMLP was used in the example here. There will also be diagnostic messages if anything unusual was detected, and specific performance details as well, some of them technical. The solver manuals will help explain these.

The solver report from the example is

```
BDMLP 1.3     Mar 21, 2001 WIN.BD.NA 20.0 056.043.039.WAT

 Originally developed by
  A. Brooke, A. Drud, and A. Meeraus,
  World Bank, Washington, D.C., U.S.A.

 Work space allocated           --    0.02 Mb

      EXIT -- OPTIMAL SOLUTION FOUND.
```

In case of serious trouble, the GAMS LST 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 manual or help from a more experienced user is recommended. This information can be standardly included by entering a line containing the statement Option sysout = on ; in the program above the solve statement.

© 2022 Prof. Bruce McCarl

3.5.6.7 The variable and equation solution listing

The next section of the LST file is an equation by equation then variable by variable listing of the solution returned to GAMS by the solver. Each individual equation and variable is listed by case under each block of equations and variables. The order of the equations and variables are the same as in the symbol listing described before.

```
---- EQU AVAILABLE  RESOURCES AVAILABLE

               LOWER     LEVEL     UPPER    MARGINAL

plantcap        -INF     10.000    10.000     6.000
salecontrct     -INF     -5.000     3.000      .


---- VAR PRODUCTION  ITEMS PRODUCED BY PROCESS

             LOWER     LEVEL     UPPER    MARGINAL

makechair      .         .        +INF    -15.000
maketable      .        5.000     +INF       .
makelamp       .         .        +INF     -6.100

                        LOWER     LEVEL     UPPER    MARGINAL

---- VAR PROFIT         -INF     60.000     +INF       .

  PROFIT  TOTALPROFIT
```

The columns associated with each entry have the following meaning,

- Block separator ----
- Equation or Variable identifier
- Lower bound (.lo)
- Level value (.l)
- Upper bound (.up)
- Marginal (.m)

Notes:

- 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 are described in the Variables, Equations, Models and Solves chapter.
- The numbers are printed with fixed precision, but the values are returned within GAMS have full machine accuracy.

© 2022 Prof. Bruce McCarl

- The single dots '.' represent zeros.
- If present 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 marginals are zero for basic variables, and nonzero for others.
- 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.
- For models that are not solved to optimality, some items may additionally be marked with the following flags.

Flag	Description
Infes	The item is infeasible. This mark is made for any entry whose level value is not between the upper and lower bounds.
Nopt	The item 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.

- 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.

3.5.6.7.1 Including slacks in the output

GAMS, unlike the rest of the mathematical programming world, includes equation "levels" in its output, not slacks. An equation level for the equation $AX \leq b$ is the term AX whereas a slack is b-AX. Users desiring slacks can get them by inserting the command below anywhere before the solve statement as is done in the example **resource.gms**. This is illustrated later in this chapter.

```
option solslack=1;
```

3.5.6.8 Ranging analysis

Some users are interested in getting ranging output in the form of LP cost and right hand side ranging results. Unfortunately, the base version of GAMS does not yield such information. The user wishing such information has two alternatives. First, one may cause the model to be repeatedly solved under a set of values for the key parameter using the procedures discussed in the Doing a Comparative Analysis with GAMS chapter but this is cumbersome if a lot of parameters are involved. Second, one can use solver dependent features of GAMS (which currently work with OSL or CPLEX) and retrieve the ranging information following the procedures discussed next. In turn the ranging information is included in the LST file and can be retrieved into a GAMS parameter.

There is a document on the GAMS web page called Sensitivity Analysis, with GAMS/CPLEX and GAMS/OSL at *https://www.gams.com/latest/docs/S_CPLEX.html#CPLEX_SensitivityAnalysis* which gives instructions on how to get ranging analyses from the OSL or CPLEX solvers. Use of this approach requires one to implement a solver options file telling the solver to generate all possible or selected ranging information. There is also an option one can use which causes the ranging information to be

© 2022 Prof. Bruce McCarl

saved in an auxiliary file importable by GAMS, subject to some potential small editing changes.

The steps to using this procedure are as follows

I. When solving an LP type of problem make sure you are using either OSL or CPLEX as the LP solver through the solver selection procedures discussed in the Variables, Equations, Models and Solves chapter.

II. Define an option file of one of two forms

- If you wish all the variables and equations to be subjected to ranging then place the following lines in the option file for CPLEX

```
objrng all
rhsrng all
```

or for OSL use

```
objrng
rhsrng
```

- If you wish only selected items to be ranged enter the commands

```
objrng variablename1,variablename2
rhsrng equationname
```

in both OSL and CPLEX where variablename and equationname are named variables and equations in your GAMS model.

  - These options can be repeated to specify ranging for more than one variable or equation.
  - One cannot nominate individual cases of a variable or equation.

III. Add lines just before the solve statement that tell GAMS to activate an options file and write a dictionary file

```
transport.OptFile=1;
transport.DictFile = 4;
```

IV. One may wish to include these items in the GAMS parameter so they may be used in calculations or report writing. A file that may be included into GAMS is automatically generated if one adds a line to the options file of the form

```
rngrestart filename
```

where filename is the name of the include file.

Example:

Suppose we take the problem **resource.gms** and set it up to do include ranging analysis. To do this we add the colored lines as below between the model statement and the solve statement creating the file **ranging.gms**.

© 2022 Prof. Bruce McCarl

```
MODEL RESALLOC /ALL/;
option lp=cplex;
FILE  OPT   Cplex option file  / cplex.OPT /;
PUT OPT;
PUT 'objrng all '/
    'rhsrng all '/;
PUTCLOSE OPT;
resalloc.optfile=1;
resalloc.dictfile=4;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
```

These lines choose the solver, write the options file using the procedure discussed in the Solver Option files chapter, activate the options file and write the dictionary file.

In turn the LST file output is augmented with the ranging information as follows

```
EQUATION NAME                    LOWER          CURRENT            UPPER
-------------                    -----          -------            -----
OBJT                              -INF              0              +INF
AVAILABLE(SMLLATHE)              95.6204           140            151.333
AVAILABLE(LRGLATHE)              83.9286            90            112.705
AVAILABLE(CARVER)                103.093           120              +INF
AVAILABLE(LABOR)                 97.9317           125            175.453


VARIABLE NAME                    LOWER          CURRENT            UPPER
-------------                    -----          -------            -----
PRODUCTION(FUNCTNORM)          -3.92507                0           27.8421
PRODUCTION(FUNCTMXSML)            -INF                 0           11.2991
PRODUCTION(FUNCTMXLRG)            -INF                 0           4.07934
PRODUCTION(FANCYNORM)          -4.97175                0           4.42299
PRODUCTION(FANCYMXSML)            -INF                 0           8.39683
PRODUCTION(FANCYMXLRG)         -4.29115                0           14.7057
PROFIT                      1.33227e-015                1             +INF
```

In our example adding the line

```
rngrestart rngfile.gms
```

to the options file (**rangeinc.gms**) causes the file **rngfile.gms** to be generated that contains the ranging results in parameters. These parameters are named with the variable and equation names with the letters RNG appended. They have the same basic set dependency but with an additional set (RNGLIM) added. That set is assumed to be defined by the user and has the elements lo and up for the upper and lower ranges. In this case the file looks like

```
PARAMETER  OBJTRNG(RNGLIM) /
LO -INF
UP +INF
/;
PARAMETER  AVAILABLERNG(RESOURCE,RNGLIM) /
```

© 2022 Prof. Bruce McCarl

```
SMLLATHE.LO          95.62037037
SMLLATHE.UP          151.3333333
LRGLATHE.LO          83.92857143
LRGLATHE.UP          112.7045827
CARVER.LO           103.0926264
CARVER.UP +INF
LABOR.LO           97.93170732
LABOR.UP           175.4526316
/;
PARAMETER  PRODUCTIONRNG(PROCESS,RNGLIM) /
FUNCTNORM.LO         -3.925065963
FUNCTNORM.UP          27.84210526
FUNCTMXSML.LO -INF
FUNCTMXSML.UP          11.29905545
FUNCTMXLRG.LO -INF
FUNCTMXLRG.UP          4.079341865
FANCYNORM.LO         -4.971748151
FANCYNORM.UP          4.422993062
FANCYMXSML.LO -INF
FANCYMXSML.UP            8.3968312
FANCYMXLRG.LO         -4.291153846
FANCYMXLRG.UP          14.70565635
/;
PARAMETER  PROFITRNG(RNGLIM) /
LO      1.33226763e-015
UP +INF
/;
```

Notes:

- The cost ranges for the variables for all but the variable maximized will always be centered on zero. This occurs because the model is of the form

  $$\begin{aligned} \text{Max} \quad & Z + 0X \\ & Z - CX = 0 \\ & AX \leq b \end{aligned}$$

  as discussed in the Quick Start Tutorial chapter.

- The parameter file of ranging results may be included immediately in the program using the GAMS to GAMS calling procedure as discussed in the Links to Other Programs Including Spreadsheets chapter or as implemented below (**rangeinc.gms** , **incmyranges.gms**)

```
 MODEL RESALLOC /ALL/;
 option lp=cplex;
 FILE  OPT   Cplex option file  / cplex.OPT /;
 PUT OPT;
$setglobal filename rngfile.gms
 PUT 'objrng all '/
     'rhsrng all '/
     'rngrestart %filename%'/;
```

© 2022 Prof. Bruce McCarl

```
PUTCLOSE OPT;
resalloc.optfile=1;
resalloc.dictfile=4;
SOLVE RESALLOC USING LP MAXIMIZING PROFIT;

Set rnglim /lo,up/;
PARAMETER  OBJTRNG(RNGLIM)
PARAMETER  AVAILABLERNG(RESOURCE,RNGLIM)
PARAMETER  PRODUCTIONRNG(PROCESS,RNGLIM)
PARAMETER  PROFITRNG(RNGLIM) ;
execute_unload 'passtorange.gdx',resource,process,rnglim;
execute 'GAMS incmyranges --filename=%filename%'
execute_load 'passtorange.gdx',OBJTRNG,
                               AVAILABLERNG,
                               PRODUCTIONRNG,
                               PROFITRNG;
```

where a couple of tricks are used

- the file name with the ranges are passed using a control variable established with a setglobal, referenced with %varnam% and the – command line parameter as discussed in the Conditional Compilation chapter.
- The sets defining the items are unloaded into a GDX file for inclusion into the GAMS program we will call to create a GDX file of results. Use of GDX commands is discussed in the Using GAMS Data Exchange or GDX Files chapter.
- GAMS is executed and the program executed is as follows

```
set resource,
    process,
    rnglim;
$gdxin passtorange.gdx
$Load resource process rnglim
$include "%filename%"
execute_unload 'passtorange.gdx',OBJTRNG,
                               AVAILABLERNG,
                               PRODUCTIONRNG,
                               PROFITRNG;
```

  where filename is passed though the – parameter in the execution
  the names of the range containing parameters must be known and the appropriate unload commands issued.

- The resultant range containing parameter names are loaded in.

- In a more complex model one may wish to only range parts of the model as illustrated by the option file below that is implemented in the context of the file **agreste.gms** by the file **rangag.gms**.

```
objrng xcrop,xlive,sales
rhsrng labc,landb
```

© 2022 Prof. Bruce McCarl

3.5.6.9 Final execution summary

The final execution summary appears next which has the results of any post solution displays plus a report on the final execution time and memory use.

Users can employ display to crate an entry in the LST file with the nonzero data for program items. The output from the display statement on line 34 of the LST file for the **shortmodel.gms** example is shown below. The format of the display statement output can be altered as discussed in the Improving Output via Report Writing chapter.

```
----     34 PARAMETER solprod  report of production
maketable 5.000
```

3.5.6.10 Report summary

The final section of the solution listing is more timing information and the report summary, marked with four asterisks (as are All important components of the output).

The time display is in the same format as that discussed above and for the example is

```
EXECUTION TIME       =        0.110 SECONDS    1.4 Mb      WIN200-121
```

The report summary 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 it the reported solution is infeasible. A domain error count is only shown if the problem is nonlinear.

```
**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
                             1  PROJECTED
```

The projected entry indicates the number of variable and equation levels that are set to bounds when their numerical value falls below the tolerance (tolproj) and their levels are set to their lower (often to zero) or upper bound.

3.5.6.11 File summary

The last piece of the LST file gives the names of files used. If work files (save or restart) have been used, they will be named here as well.

```
**** FILE SUMMARY

INPUT     C:\GAMS\GAMSPDF\SHORTMODEL.GMS
OUTPUT    C:\GAMS\GAMSPDF\SHORTMODEL.LST
```

3.5.7 Managing output pages

Pages may be managed adding titles, ejecting pages, and managing width and length.

© 2022 Prof. Bruce McCarl

Page width and height
New pages
Adding an output title to each page

3.5.7.1 Page width and height

Users may wish to exercise control over page width and length. The default page length is 60 lines. This often causes longer files to contain a lot of GAMS headers in inconvenient spots. One can respecify page size in several ways.

- Job specific page characteristics are specified on the GAMS call using the general syntax

  ```
  GAMS modelname ps=n1 pw=n2
  ```

  where ps gives the page length and can be as small as 30 or as big as 9999 and pw the page width which must be between 72 and 32767.

  More specifically

  ```
  GAMS trnsport ps=9999 pw=100
  ```

- The default page length and width can be customized by editing a file in the GAMS source directory as discussed in the Customization chapter. In particular on Windows 95/98 machines the file is called GMSPRM95.txt (Note on NT and UNIX/LINUX machines it has a slightly different name -- gmsprmnt.txt and gmsprmun.txt respectively). In this file one defines the page length with a line ps followed by a space and a number

  ```
  ps 1000
  ```

  and page width with

  ```
  pw 100
  ```

- In the IDE job specific page characteristics are specified in the GAMS execution parameter box in the upper right hand corner adding

  ```
  ps=9999 pw=100
  ```

  where ps gives the page length and can be as small as 30 or as big as 9999 and pw the page width which must be between 72 and 32767.

- In the IDE characteristics for all jobs are specified using the file options menu choice under the output tab in the boxes for page width and page height (same as page size above). The page height and can be as small as 30 or as big as 9999 and the page width which must be between 72 and 32767.

3.5.7.2 New pages

Users may wish to cause new pages in the echo print to begin unilaterally or if there are only a few lines left on the page using the commands

$Eject	Dollar command which starts a new page in LST file
$Lines	Dollar command which starts new page if less than n lines are left on a page

© 2022 Prof. Bruce McCarl

3.5.7.3 Adding an output title to each page

A title and subtitle can be placed on each and every page using

$Stitle	Defines subtitle for LST file
$Title	Defines LST file title

For example in the Model library file **co2mge.gms** at various points the following appear

```
$TITLE  Carbon-Related Trade Model (static)  (CO2MGE,SEQ=142)
$stitle: CARBON-RELATED TRADE MODEL - BENCHMARK REPLICATION
$stitle:CARBON-RELATED TRADE MODEL - OECD ABATEMENT WITH HIGH LEAKAGE
```

which results in every page of the LST file being started with the line

```
Carbon-Related Trade Model (static)  (CO2MGE,SEQ=142)
```

and pages after the $stitle having pages that start with

```
Carbon-Related Trade Model (static)  (CO2MGE,SEQ=142)
: CARBON-RELATED TRADE MODEL - BENCHMARK REPLICATION
```

until the next stitle where the page heading becomes

```
Carbon-Related Trade Model (static)  (CO2MGE,SEQ=142)
:CARBON-RELATED TRADE MODEL - OECD ABATEMENT WITH HIGH LEAKAGE
```

3.5.8 Managing output volume

Sometimes the GAMS output volume can be overwhelmingly large. There are several actions one can undertake to limit the output. These are listed below:

- Eliminate model listing
- Eliminate cross reference map
- Eliminate symbol list
- Eliminate solution output
- Eliminate echo print
- Restrict output just to a few displays

3.5.8.1 Eliminate model listing

To limit the model listing obtained set Limrow and Limcol to zero as follows

```
Option Limrow=0;
Option Limcol=0;
```

© 2022 Prof. Bruce McCarl

3.5.8.2 Eliminate cross reference map

To eliminate the cross-reference map use the command

```
$Offsymxref
```

Note that this is eliminated by default in the IDE.

3.5.8.3 Eliminate symbol list

To eliminate the alphabetic listing of the symbol table use the command.

```
$Offsymlist
```

Note that this is eliminated by default in the IDE.

3.5.8.4 Eliminate solution output

To eliminate the solver generated solution output use the command

```
Option Solprint=off;
```

3.5.8.5 Eliminate echo print

To eliminate listing of segments of code use the command $Offlisting. Note anything placed between the following two commands will not be copied to the LST file

```
$OFFLISTING
text in between will not appear in the LST file
$Onlisting
```

3.5.8.6 Restrict output just to a few displays

One can also use a save restart strategy where the LST file just contains the desired display statements. This allows one to only concentrate on a narrow set of output while remaining capable of generating a lot more output. This strategy is discussed in the Saves and Restarts chapter.

3.5.9 Adding slack variables to the output

By default GAMS output contains equation levels. An equation level is the activity on the left-hand side of an equation. Most people are not taught such a concept but rather expect to see slack variables. GAMS will replaces the equation levels with slack variables in the output if one uses the option command:

```
Option solslack=1
```

Without this command when solving **slack.gms** the equation part of the solution is

© 2022 Prof. Bruce McCarl

```
---- EQU AVAILABLE   RESOURCES AVAILABLE
                LOWER     LEVEL     UPPER    MARGINAL
Land             -INF   800.000   800.000   118.700
Spring Labor     -INF  3114.000  4800.000       .
Fall Labor       -INF  3300.000  3300.000     4.600
```

but with it

```
                LOWER     SLACK     UPPER    MARGINAL
Land             -INF       .     800.000   118.700
Spring Labor     -INF  1686.000  4800.000       .
Fall Labor       -INF       .     3300.000     4.600
```

3.5.10 Sending messages to the LOG file

A message may be sent to the LOG file using the command $LOG. It is employed using the syntax

```
$LOG text to send
```

or by using Put commands as discussed in the Output via Put Commands chapter.

3.6 Writing Models and Good Modeling Practices

The style with which one formats ones GAMS model is a matter of both needed adherence to language requirements and individual preference. In this chapter I discuss the individual preference aspects. In particular, I will assert the virtues of my preferences but obviously others can choose their own practices.

Formatting models - my recommendations

3.6.1 Formatting models - my recommendations

Users have many options on how they could present a GAMS model. Over more than 15 years of GAMS experience including the relative frequently event where I need to work with models that others have constructed I have arrived at a number of practices I try to follow.

In general I feel that it is in a modelers hands as to how much self-documentation a piece of GAMS code contains. I do feel there are purposeful actions that can improve documentation and will explain a number of practices that could be followed here. The extent to which such practices are followed often determines how easy it is to reuse or repair a model at a later time or how easily a colleague (or in my case a consultant) can work with that code.

The main categories of actions that are possible are

- Use longer names and descriptions
- Include comments on procedures and nature and sources of data
- Include as much raw data as possible as opposed to externally calculated data
- Don't use an * as a set specification for input data
- Use sets to aid in readability
- Format files to improve model readability

© 2022 Prof. Bruce McCarl

- Use some other possible conventions

Below I cover each of these.

3.6.1.1 Use longer names and descriptions

One can radically affect the readability of a piece of GAMS code by using longer self explanatory names of parameters, sets etc. in that code. To illustrate the difference this might make let us look at the example **robert.gms** from the GAMS model library. A part of that model is reproduced below

```
Variables   x(p,tt)  production and sales
            s(r,tt)  opening stocks
            profit
 Positive variables x, s;
 Equations  cc(t)    capacity constarint
            sb(r,tt) stock balance
            pd       profit definition ;
 cc(t)..     sum(p, x(p,t)) =l= m;
 sb(r,tt+1).. s(r,tt+1) =e= s(r,tt) - sum(p, a(r,p)*x(p,tt));
 pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
               - sum(r, misc("storagec",r)*s(r,t)))
               + sum(r, misc("resvalue",r)*s(r,"4"));
 s.up(r,"1") = misc("max-stock",r);
```

After an hour of so of looking at the model I reformatted it as follows (**good.gms**)

```
 Variables   production(process,Quarters)  production and sales
            openstock(rawmateral,Quarters)  opening stocks
            profit ;
 Positive variables production, openstock;
 Equations  capacity(quarter)              capacity constarint
            stockbalan(rawmateral,Quarters) stock balance
            profitacct                     profit definition ;
 capacity(quarter)..
        sum(process, production(process,quarter)) =l= mxcapacity;
 stockbalan(rawmateral,Quarters+1)..
      openstock(rawmateral,Quarters+1) =e=
      openstock(rawmateral,Quarters)
      - sum(process, usage(rawmateral,process)
                   *production(process,Quarters));
 profitacct.. profit =e=
   sum(quarter,
       sum(process, expectprof(process,quarter)
                    *production(process,quarter))
     - sum(rawmateral, miscdata("store-cost",rawmateral)*
                       openstock(rawmateral,quarter)))
     + sum(rawmateral, miscdata("endinv-value",rawmateral)
                      *openstock(rawmateral,"winter"));
 openstock.up(rawmateral,"spring") = miscdata("max-stock",rawmateral);
```

© 2022 Prof. Bruce McCarl

Note these two models do the same thing but different longer names are used in the second. Also note that for example instead of using the set name tt I use Quarters and instead of calling the variables x I call it production. The question is which tells you more and which would you want to face in 5 years?

Lets look at some more from **robert.gms**. Another segment of code is

```
Sets  p     products       / low, medium, high /
      r     raw materials  / scrap, new /
      tt    long horizon   / 1*4 /
      t(tt) short horizon  / 1*3 /
Table  a(r,p)  input coefficients
          low  medium  high
 scrap      5     3      1
 new        1     2      3
Table  c(p,t)  expected profits
            1    2    3
 low       25   20   10
 medium    50   50   50
 high      75   80  100
```

This can be reformatted to become (**good.gms**)

```
 Sets  process   production processes available
               / low uses a low amount of new materials,
                 medium uses a medium amount of new materials,
                 high uses a high amount of new materials/
       rawmateral   source of raw materials  / scrap, new /
       Quarters    long horizon  / spring, summer, fall ,winter /
       quarter(Quarters) short horizon  / spring, summer, fall /
 Table  usage(rawmateral,process)  input coefficients
          low  medium  high
 scrap      5     3      1
 new        1     2      3
 Table  expectprof(process,quarters)  expected profits
         spring summer fall
 low        25    20    10
 medium     50    50    50
 high       75    80   100
```

Here I use longer names for the set elements, the sets themselves along with explanatory text documenting the definition of the set elements. So again the question is which one makes more sense to you? I obviously feel the latter.

3.6.1.1.1 Basic point

One can increase documentation be lengthening the names used in GAMS code. In particular

- GAMS allows 63 character long names and 255 characters of explanatory text defining each

© 2022 Prof. Bruce McCarl

of the following items: sets, parameters, variables, scalars, acronyms, equations, models and files.

I feel users should exploit this and use descriptive 63 character names.

X is a wonderful name in theory, yet lousy in practice, as is A(i,j).

Type a little more and reap the rewards in being able to figure out models later. More typing always helps and it is just not that hard.

More than half the models in the GAMS model library have such failings.

- Let no item go undefined. Enter explanatory text or comments containing units, sources and descriptions.
- Check for completeness in the symbol listing with $Onsymlist making sure all names are somewhat apparent and all items have explanatory text as illustrated in the Standard Output chapter.
- Associate text with set element definitions and use the up to 63 character set element name capability. (Note that names longer that 10 characters do not work well in multi column displays.)

Remember only **You** can cause your GAMS code to be self documenting.

3.6.1.2 Include comments on procedures and data nature and sources

Procedures for documentation and clarity may also be employed in setting up the model and the data therein. Questions I ask when looking at a set of model data are:

- Why was a constraint set up in the way it is implemented?
- What are the units on the variables and equations?
- Where did the data come from?
- What are the characteristics of the data such as units, and year of applicability?

Such questions often apply to segments of GAMS code. Frequently it's nice to add in descriptions identifying assumptions, intent of equation terms, data sources including document name, page number, table number, year of applicability, units, URL etc. You can do this with comments (* in column 1) statements, $Ontext $Offtext sequences or inline comments as illustrated in **commentdol.gms** or as discussed in the Including Comments chapter as follows

```
* this is a one line comment that could describe data
$Ontext
My data would be described in this multiline comment
This is the second line
$Offtext
*I could tell what the equation below is doing
x = sum(I,z(i)) ; # this is an end of line comment
x = sum(I,z(i)) ; { this is an inline comment } r=sum(I,z(i)) ;
```

Note inline comments must be enabled with the commands $inlinecom or $eolcom as in the following two statements

© 2022 Prof. Bruce McCarl

```
$eolcom #
$inlinecom {}
```

3.6.1.3 Entering raw versus calculated data

Modelers often face two choices with respect to data.

- Enter raw data into GAMS and transform it to the extent needed inside GAMS.
- Externally process data entering the final results in GAMS.

The latter choice is often motivated by the availability of the raw data in a spreadsheet where those data could be manipulated before being put into GAMS.

My recommendation (as one who often updates models and needs to figure out the embodied assumptions) is put data in as close to the form it is collected into GAMS and then manipulate that data in the GAMS code.

Why? Over time spreadsheets and other data manipulation programs change, or get lost. Also often internal documentation in such programs is weak. Putting it in GAMS keeps it all together.

3.6.1.4 Avoiding use of * in input data set specification

One can use an * to allow the universal set to be employed in an input data set position as done in the GAMS library code **robert.gms** as follows

```
Table  misc(*,r)  other data
            scrap  new
 max-stock    400  275
 storage-c     .5    2
 res-value     15   25
...
pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
              - sum(r,misc("storage-c",r)*s(r,t)))
              + sum(r,misc("res-value",r)*s(r,"4"));
```

This tells GAMS anything goes in the first index position of misc suppressing domain checking. However, if we mistyped "res-value" as "res-val" GAMS compiles and executes without error but I have lost my data and perhaps created a garbage result as in **robert2.gms**.

```
pd2.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
              - sum(r,misc("storage-c",r)*s(r,t)))
              + sum(r,misc("res-value",r)*s(r,"4"));
```

I can fix this by entering another set and altering the table definition

```
Set miscitem misc. input items
                        /res-value, .../;
      Table  misc(miscitem,r)  other data
```

© 2022 Prof. Bruce McCarl

where GAMS with the code in **Robert2.gms** now included in **robert3.gms** gives error messages for the misspelling.

So don't use * for set references in input data specifications. I learned this at the school of hard knocks when a research associate used such an input strategy and had some small typing errors that caused important data to be ignored.

3.6.1.5 Making sets work for you

Sets are used to address a family of similar items. You need to decide when to use a single versus multiple sets. There are cases when it is convenient to have an element in such a family and cases when it is not.

Suppose we have three grades of oil and three processes to crack it. Should we have one set with nine entries or make the item two-dimensional. I would do the latter. Suppose we have a budget using fertilizer, seed, labor by month and water by month. Do we have a set with 26 items or two sets one with fertilizer, seed, labor and water and the other with month names plus the label annual. I would do the latter. I err on the side of being more extensive with set definitions.

Sets should contain items treated similarly in the problem (i.e., resources like fertilizer, seed, and energy), but when there are two items crossed (i.e., monthly availability of land, labor, and water involves month and resource) one should have two sets.

3.6.1.6 Making subsets work for you

One other set definition consideration involves the use of subsets. Sometimes it is desirable to have items that can be treated simultaneously in some places, but separately elsewhere. For example, when entering crop budgets one might wish to enter yield along with usage of inputs, land, labor, and water, in one spot yet treat those differently elsewhere (i.e., where variable inputs might be in one equation, yield balance in another, with water and labor availability in yet a third and fourth equation). Subsets allow this. Consider two models

In the **Egypt.gms** model from the GAMS model library we have yields and input costs in two tables some 50 lines apart

```
Table yield (c,r)  yield for different commodites
             u-egypt   m-egypt    e-delta    m-delta     w-delta
*               ton       ton        ton        ton         ton
   wheat       1.29      1.36       1.39      1.404        1.36
   barley      1.41      1.26       1.33       .984         .96


Table cropdat(c,*) seed protein starch misc costs and pestic data
         protein  starch  seed    misc    pest    n-fer    p-fer
*          %        %      ton     le      le      ton      ton
wheat      .1     23.3    .075    12.0            0.054    0.015
barley     .1     23.3    .060     8.0            0.045    0.015
```

In my ASM model (**asmall10.gms**, **asmcrop10.gms**) I have

© 2022 Prof. Bruce McCarl

```
TABLE CCCBUDDATA(ALLI,SUBREG,CROP,WTECH,CTECH,TECH)  REGIONAL CROP BUDGET
               northeast.corn.DRYLAND.BASE.0
corn                   115.87
CROPLAND                 1.00
LABOR                    4.34
nitrogen                24.69
potassium               15.17
phosporous               7.34
```

Where everything for a crop budget is together in concurrence with practices in data sources. We use subsets (**asmsets10.gms**) to unravel the data

```
SET ALLI                ALL BUDGET ITEMS
      /  corn      ,    soybeans  ,
         cropland  ,   pasture   ,     labor,
         nitrogen  ,    potassium ,    phosporous,    trancost /
set PRIMARY(ALLI)       PRIMARY PRODUCTS
       /  cotton    ,    soybeans /
set INPUT(ALLI)    NATIONAL INPUTS
       /  nitrogen  ,    potassium ,  trancost  ,    phosporous/
set LANDTYPE(ALLI)         LAND TYPES
       /  cropland  ,    pasture   /
```

Use of subsets and a general set like ALLI allow one to both organize the input according to convenience with data sources and then deal with it efficiently in the model and report writer statements. You can also avoid the * in the input data sets.

3.6.1.7 Formatting the typing of files to improve model readability

I think there are things you can do to improve the readability as you type. A list of such practices follows:

- Enter the code in a fixed order by the following sections and keep the section together such as follows
  - Data related sets
  - Data definitions organized by type of data possibly intermixed with calculations
  - Variable and equation definitions
  - Algebraic equation definitions
  - Model and solve
  - Report writing sets and parameter definitions
  - Report writer calculations
  - Report writer displays
- Keep the sections together

© 2022 Prof. Bruce McCarl

- Format the code for readability using spacing and indents as illustrated below
  - Align item names, descriptions and definitions
  - Indent in sums, loops and ifs to delineate terms. Through indentation, and closing parentheses formatting you can reveal structure.
  - Use blank lines to set things off
  - Don't split variables between lines in equations, but rather keep them together with all their index positions.

By follow such typing practices you can make a file more readable using spacing, indents, and set labels

Case A (**badtype.gms**)

```
Sets  products  available production process / low uses low new materials
medium uses medium new materials, high uses high new materials/
rawmateral    source of raw materials  / scrap, new /
Quarters    long horizon   / spring, summer, fall ,winter /
quarter(Quarters) short horizon  / spring, summer, fall /
Scalar mxcapacity maximum production capacity/ 40 /;
Variables  production(products,Quarters)  production and sales
openstock(rawmateral,Quarters)  opening stocks, profit ;
Positive variables production, openstock;
Equations  capacity(quarter)     capacity constraint,
profitacct.. profit =e= sum(quarter, sum(products, expectprof(
products,quarter) *production(products,quarter))-sum(
rawmateral,miscdata("store-cost",rawmateral)*openstock(rawmateral
  ,quarter)))+ sum(rawmateral, miscdata("endinv-value",rawmateral)  *openstock(
```

Case B (**better.gms**)

```
Sets  products      available production process
                   / low uses a low amount of new materials,
                     medium uses a medium amount of new materials,
                     high uses a high amount of new materials/
      rawmateral    source of raw materials  / scrap, new /
      Quarters    long horizon   / spring, summer, fall ,winter /
      quarter(Quarters) short horizon  / spring, summer, fall /
 Variables  production(products,Quarters)  production and sales
            openstock(rawmateral,Quarters)  opening stocks
            profit ;
 Positive variables production, openstock;
 Equations  capacity(quarter)     capacity constarint
            stockbalan(rawmateral,Quarters) stock balance
            profitacct              profit definition ;
 profitacct..
    profit =e=
    sum(quarter,
```

© 2022 Prof. Bruce McCarl

```
          sum(products, expectprof(products,quarter)
                       *production(products,quarter)
              )
         -sum(rawmateral, miscdata("store-cost",rawmateral)*
                          openstock(rawmateral,quarter)
              )
          )
       +sum(rawmateral, miscdata("endinv-value",rawmateral)
                       *openstock(rawmateral,"winter")
          )
      ;
```

You may like case A, I don't. I find B much more readable. This matters when you have 1000 lines or more. (**asmmodel.gms**)

3.6.1.8 Other possible conventions

In addition one may develop a number of other conventions. Paul Leiby at Oak Ridge National Laboratory sent me the following

- Establish some convention (any convention) on the use of upper and lower case letters. GAMS may be case insensitive, but it will preserve your use of case for documentation purposes. I usually use lower case for text, comments, and variable descriptors, and upper case for GAMS reserved words and variable and parameter names.
- In declaring variables and parameters, always indicate the units in the variable descriptor: ie instead of

  ```
  PARAMETER VEHSALES(r)  REGIONAL VEHICLE SALES
  ```

  use

  ```
  PARAMETER VEHSALES(r)  "Regional vehicle sales ($ millions/yr)"
  ```

  This will cause the units to be shown with every DISPLAY statement for that parameter or variable, and can be a time-saver when interpreting results or entering data.
- Make a habit of always surrounding the explanatory text with quotes

  Allowing the use of special characters, such as "$", "-" or "&"

  Applying a distinct separator for the Descriptor field, demarking it and emphasizing that it is a text string
- Maintain a file modification log at the top of each file (Modification date, version number, modification made, and by whom.

With respect to the last suggestion above such a log can be entered in GAMS statements as is done in the *FASOM model* using a set specified like

```
version(*,*,*,*)
```

and accompanied by commands in the code like

© 2022 Prof. Bruce McCarl

```
version("filename","may","19","2002")=yes;
```

then in final model I display the set version so it shows me the time of alterations in the various model components and allows me to make sure my version is synchronized with others.

```
display version;
```

One other suggestion

- In assigning item names consider adopting a convention starting all sets with s_, all data with d_, all variables with v_, all equations with e_ etc.

4 Changing licenses

During the installation process you may have been prompted to point to a license file and if this was done successfully then skip this step.

4.1 Licenses on IDE

If a new license is obtained or you have alternative license files you can use operating system or IDE facilities to update your license. In the IDE choose the file options choice then the Licenses tab. There point to the License file desired using the box with three buttons in it and choose the file nominally called gamslice.txt that is located somewhere else on your computer. However a word of caution is in order. This does not copy the license file to the GAMS system directory and it is inadvertently moved or deleted in the other location it will no longer provide a license. It is thus advised that you do not use this procedure for your base license and rather copy it to the proper spot as discussed immediately below.

4.2 Licenses outside of IDE—Windows and Unix/Linux

License files do need to be updated and sometimes altered as solvers are added and maintenance paid. While this usually involves installation of a newer version of GAMS there are times when that will not be the case. In order to install a newly obtained license file one should recognize that

- This file is ordinarily named gamslice.txt
- The file needs to be copied into the GAMS system directory wherever that might be. On PCs today this is C:\program files\.GAMS227 but maybe another location dependent on GAMS vintage, country and users installation.

In turn given the contents then one needs to create and save or copy that file into the GAMS system directory with the name gamslice.txt.

This is a crucial file as it identifies what solvers the user has access to and allows upgrading of versions as long as the expiration date on the license file is after the release date of a newer version of GAMS. Note a procedure called Checkver available from *GAMS Corporation* allows one to check on license file status and possible updates.

© 2022 Prof. Bruce McCarl

5 Running Jobs with GAMS and the GAMS IDE

GAMS is a two pass program. One first uses an editor to create a file nominally with the extension GMS which contains GAMS instructions. Later when the file is judged complete one submits that file to GAMS. In turn, GAMS executes those instructions causing calculations to be done, solvers to be used and a solution file of the execution results to be created.

Two alternatives for submitting the job are discussed herein: the traditional command line approach and the IDE approach.

Basic approaches to GAMS usage
Running GAMS from the command line
Steps to using IDE

Material is then presented on use of the IDE approach.and a number of features of the IDE

IDE concept and usage
Selected techniques for use of the IDE
Unraveling complex files: Refreader
Finding out more through help
Accessing documentation outside the IDE
Saving and Using a Script
When is it not worth using?
Employing command line parameters
A difficulty you will have using IDE

Finally some material is presented on GAMS and IDE installation

IDE Installation

5.1 Basic approaches to GAMS usage

GAMS was developed well before widespread availability of graphical interfaces like Windows. Consequently, GAMS was designed to run from an operating system command line. Today GAMS may also be run on Windows machines using the so called IDE or Integrated Development Environment. Here we cover both.

5.2 Running GAMS from the command line

The basic procedure involved for running command line GAMS is to create a file (nominally with the extension gms nominally myfilename.gms where myfilename is whatever is a legal name on the operating system being used) with a text editor and when done run it with a DOS or UNIX or other operating system command line instruction like

```
GAMS tranport
```

where **tranport.gms** is the file to be run. Note the gms extension may be omitted and GAMS will still find the file.

© 2022 Prof. Bruce McCarl

GAMS would then run the job and in turn create a LST file (tranport.LST in this case) of problem results. One would then edit the LST file to find any error messages, solution output, report writing displays etc and reedit the gms file if there were need to fix anything or alter the model contents.

The basic command line GAMS call also allows a number of arguments as illustrated below

```
GAMS TRNSPORT pw=80 ps=9999 s=mysave
```

which sets the page width to 80, the page length to 9999 and saves work files. The full array of possible command line arguments is discussed in the GAMS Command Line Parameters chapter.

5.3 IDE concept and usage

Today with the average user becoming oriented to graphical interfaces it was a natural development to create the IDE. The IDE is a GAMS Corporation product providing an Integrated Development Environment that is designed to provide a Windows graphical interface to allow for editing, development, debugging, and running of GAMS jobs all in one program. Specifically the IDE

- Contains a fully featured text editor.
- Has knowledge of some GAMS syntax changing the color of displays to reveal aspects of GAMS statements.
- Has a direct interface to GAMS permitting one to run jobs from within the IDE.
- Has error discovery procedures addressing exact places in source files where compilation errors arise.
- Has a facility to automatically open the GAMS LST, LOG, PUT, and GDX output files.
- Allows customization of GAMS command line options either on a one time basis or for all jobs run with the IDE.
- Facilitates some installation and maintenance tasks.
- Allows one to view GDX files as discussed int the Using GAMS Data Exchange or GDX Files chapter.

The remaining part of this chapter is devoted to introducing users to IDE usage and features.

Steps to using IDE
Working with your own file
Fixing compilation errors
Selected techniques for use of the IDE
Finding out more through help
Unraveling complex files: Refreader
Employing command line parameters
A difficulty you will have using IDE
When is it not worth using?

© 2022 Prof. Bruce McCarl

5.3.1 Steps to using IDE

The use of the IDE after installation involves a multi-step process:

I. Create a project by going to the file selection in the upper left corner.

II. Define a project name and location.

III. Create or open an existing file of GAMS instructions.

IV. Prepare the file so you think it is ready for execution.

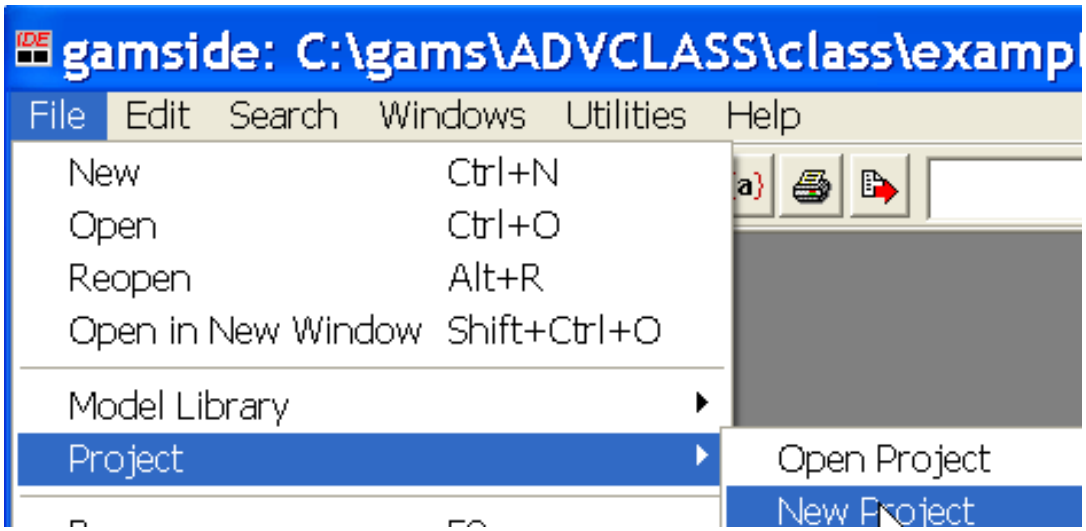
V. Run the file with GAMS by clicking the run button or pressing F9.

VI. Open and navigate around the output.

Each of these steps is discussed below assuming you have first opened the IDE through the icon, start menu or Explorer.

5.3.1.1 Create a project

Open the File menu choice. Select Project and New project (Later you will use your previous projects).



What is a project? The IDE employs a "Project" file for two purposes.

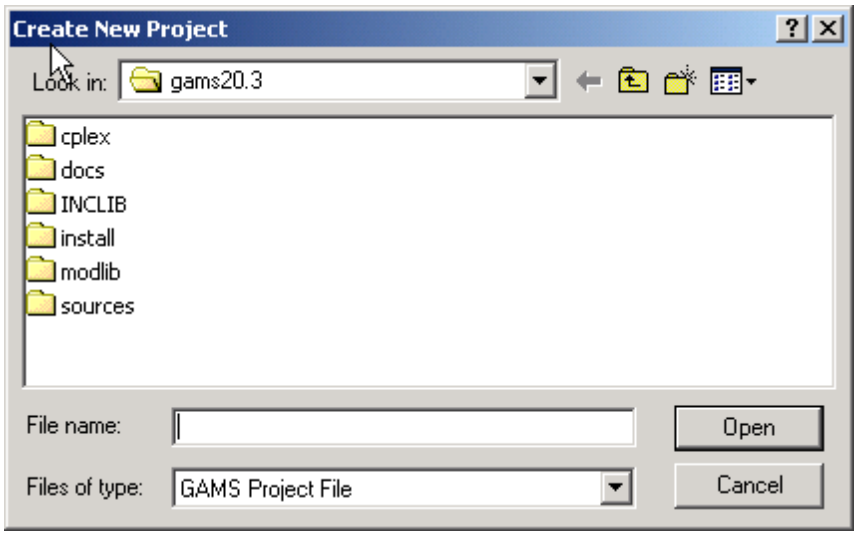
- The project location determines where all saved files are placed (to place files elsewhere use the save as dialogue) and where GAMS looks for files when executing.
- The project saves file names and program options associated with the effort in a file called projectname.gpr.

It is a good idea to define a new project every time you wish to change the file storage directory.

5.3.1.1.1 Defining a project name and location.

Locate the project in a directory you want to use. All files associated with this project will be saved in that directory.

© 2022 Prof. Bruce McCarl

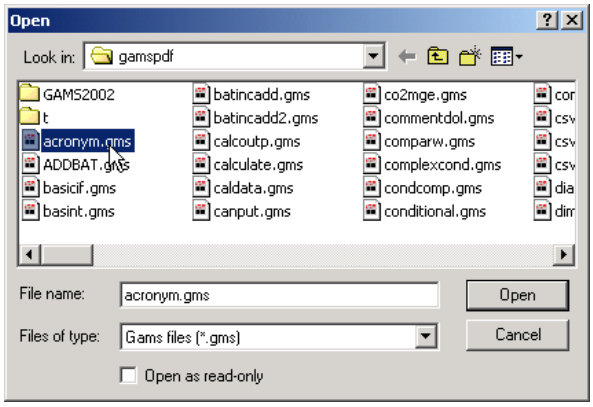


In the "File name" area enter in a name for the project file you wish to use. This defines the directory where for the most part the files you wish to access (those specified without full file path names) are located. If we were doing this for a project on hydropower, we would go to a suitable subdirectory and create a subdirectory called hydropower and name the project hydropower. In turn, a file called hydropower.gpr will be created in that directory and will store all project information. The extension gpr stands for GAMS project.

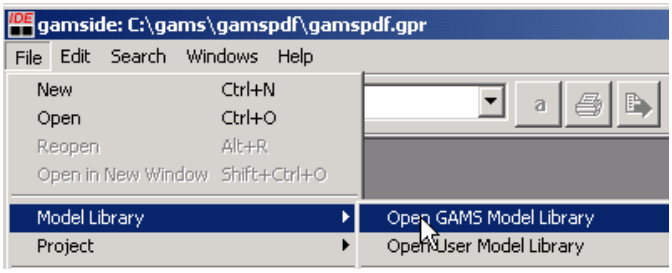
5.3.1.1.2 Creating or opening an existing GMS file

Now we want to work with an input file. Several cases are possible for the source of this file

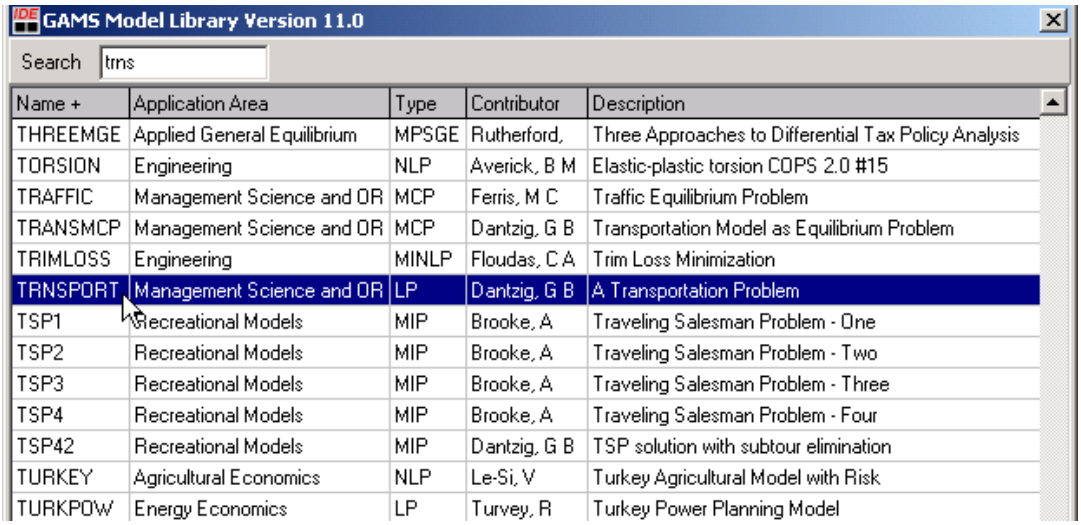
- You can create a new file
- You can open an existing file using the file open dialogue



- You can open a GAMS model library file using the file Model library dialogue



The last is the simplest and the one we illustrate first. Select a model like **trnsport.gms** by scrolling down or typing the name trnsport in the search box until it is the chosen one

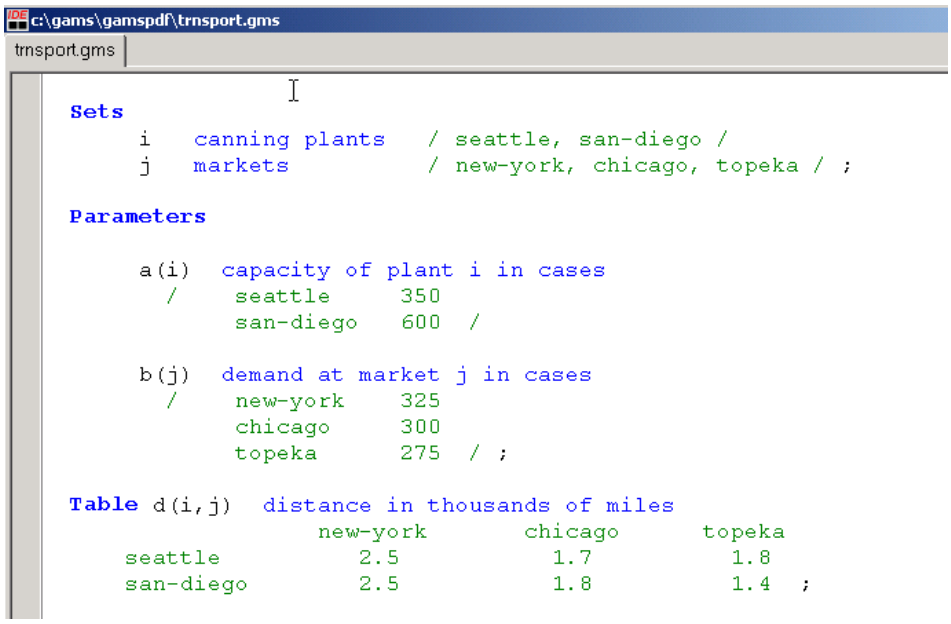


In turn this file will be automatically saved in your project directory (this is the directory where the project file is located). *Note outside the IDE this is done using the command line instruction gamslib modelname or in this case gamslib trnsport.*

5.3.1.2 Preparing file for execution

When using model library **trnsport.gms** should now appear as part of your IDE screen.

© 2022 Prof. Bruce McCarl



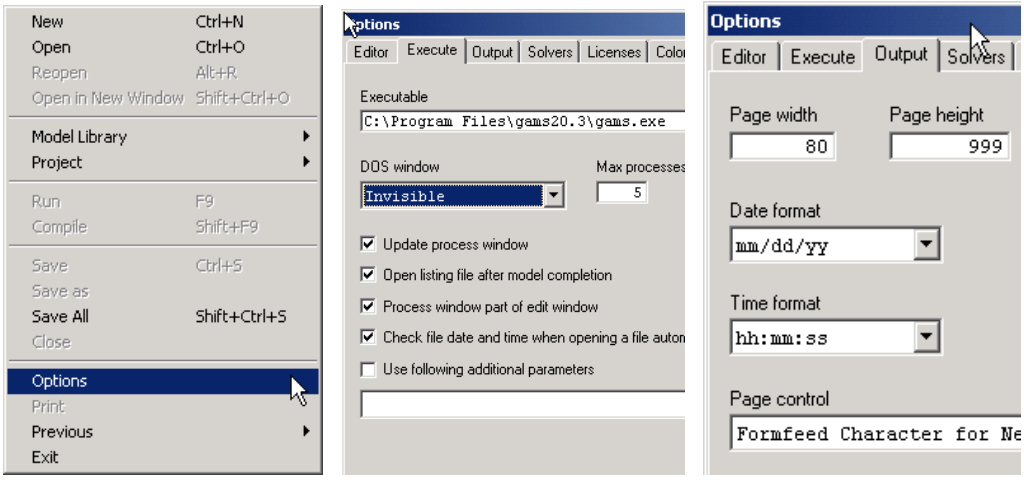
The IDE contains a full-featured editor. Go through the file and change what you want. All of the commands for the editor are documented in the help option of the IDE while some selected ones are highlighted below.

5.3.1.3 Select default IDE functions

The IDE allows one to personalize its function to user desires. Here we cover a few such options.

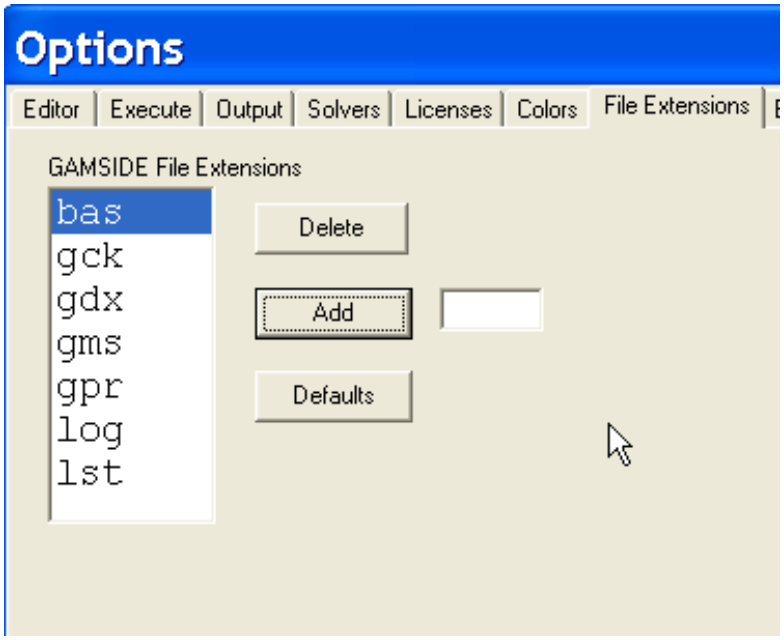
5.3.1.3.1 Page size and LST file opening

You can use the File and Options dialogue to set default page width and length. Namely select File and Options then the Output tab where you can set the page length (I use 9999). You can also use the File and Options Execute tab to cause to automatically open the LST file by making sure there is a check in the box for update process window.



5.3.1.3.2 Make IDE the default GMS file processor

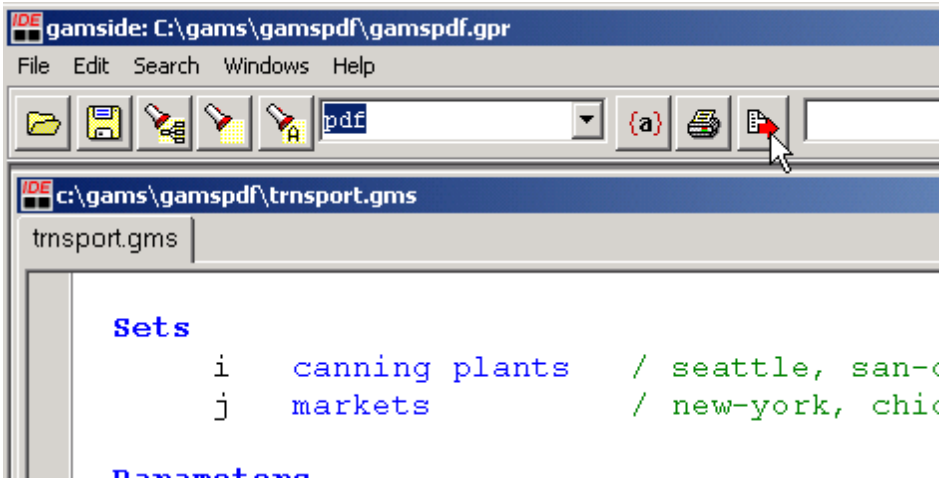
When one clicks on a file in the explorer or in this document it only will open up a file if a default program is associated with it. You can make the IDE the file that is activated when you click on a GMS ,LST , LOG or PUT. You do this by starting up the File and Options dialogue and select the File Extensions tab



then you enter GMS in small box on the right and click on Add or press the Defaults button which will add GMS, GDX, LOG and LST files.

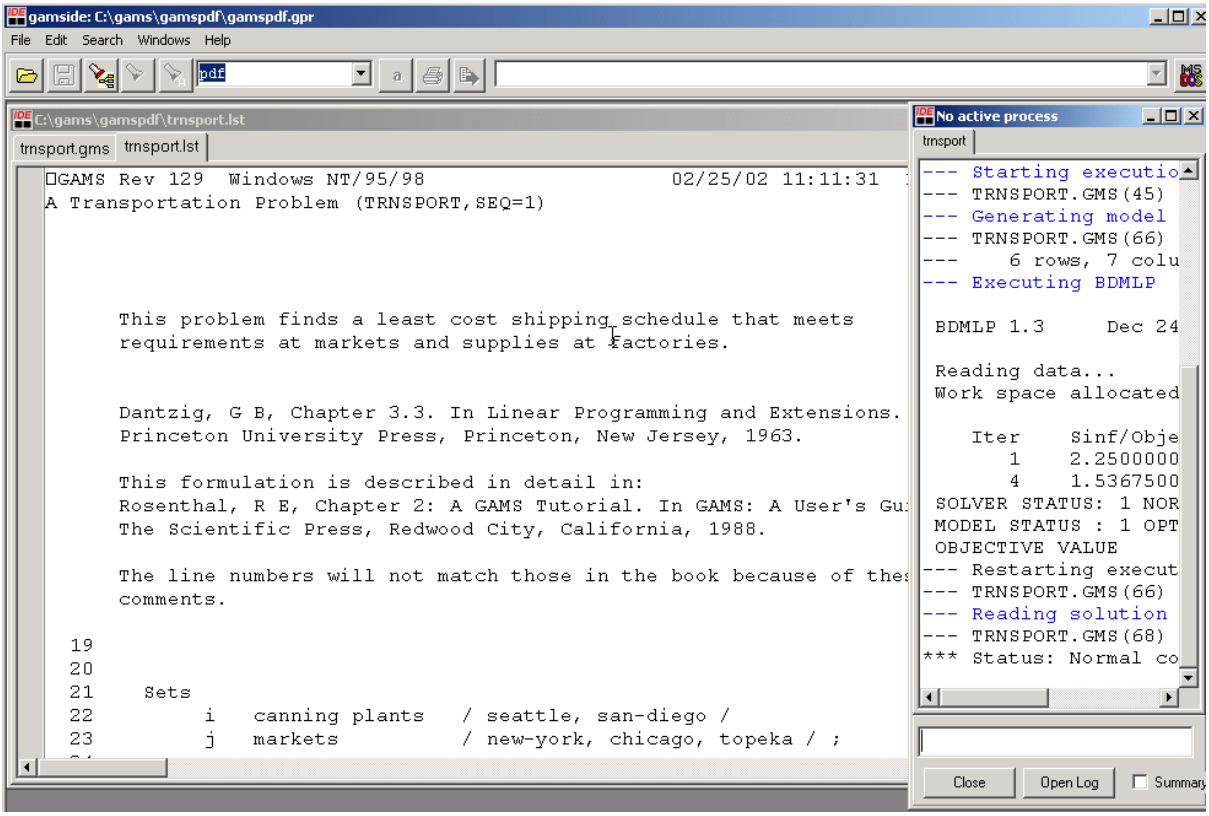
5.3.1.4 Run GAMS by clicking the run button

Now how do we run GAMS. We do this by either clicking with the mouse on the button that contains a red arrow or pressing the F9 key.



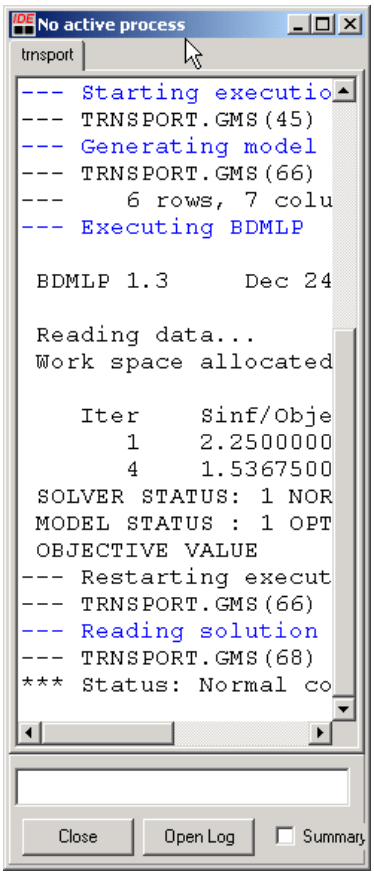
© 2022 Prof. Bruce McCarl

In turn this causes GAMS to run and also causes the so-called process window to appear which gives a LOG (actually containing the LOG file) of the steps GAMS goes through in running the model.



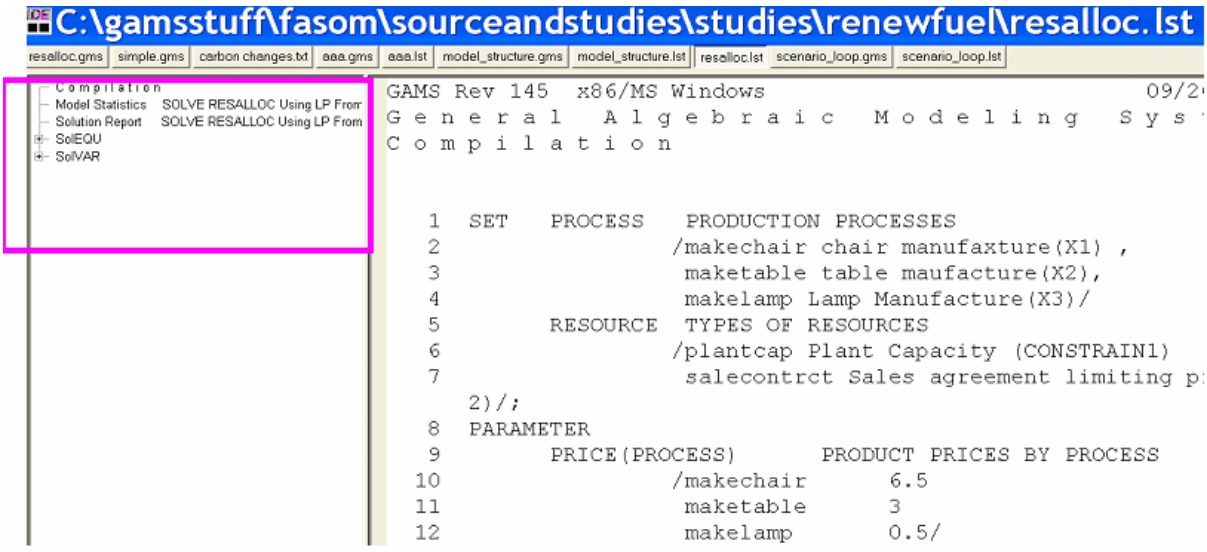
One may also have to do some housekeeping to get the process window so it does not obscure the LST file. I recommend narrowing it and pulling it off to the right as shown below, but users may place it at the bottom, maximize it or do whatever appeals.

© 2022 Prof. Bruce McCarl



5.3.1.5 Open and navigate around the output

There are two ways to do this first one may use the process window (on the right below outlined in pink) or one may use the LXI navigation aid (on the left below outlined in red) that is attached to the LST file.



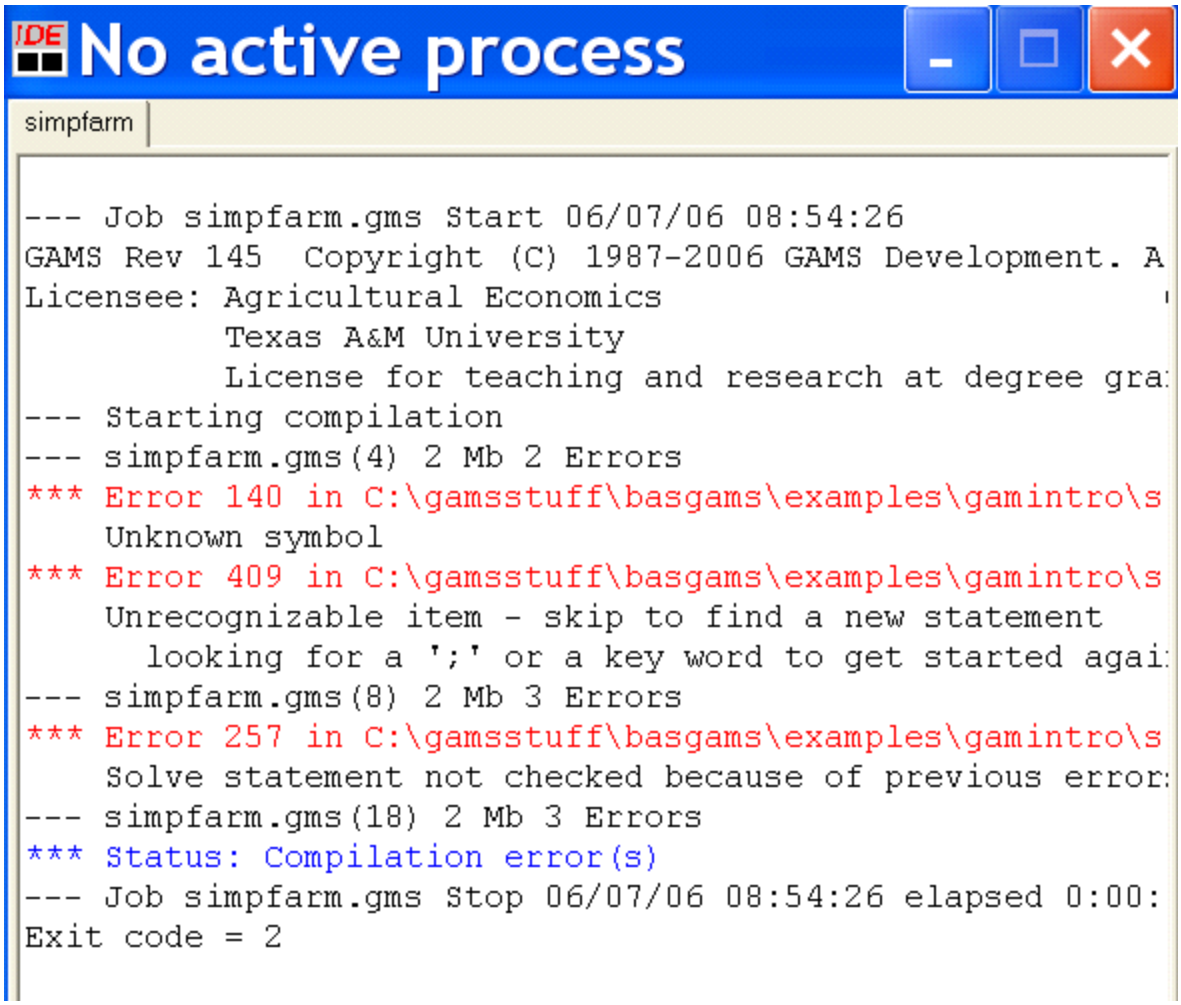
Using the Process Window

© 2022 Prof. Bruce McCarl

Using the LST file navigation window

5.3.1.5.1 Using the process window

By double clicking on lines in the process window you can access program output both in general and at particular locations.

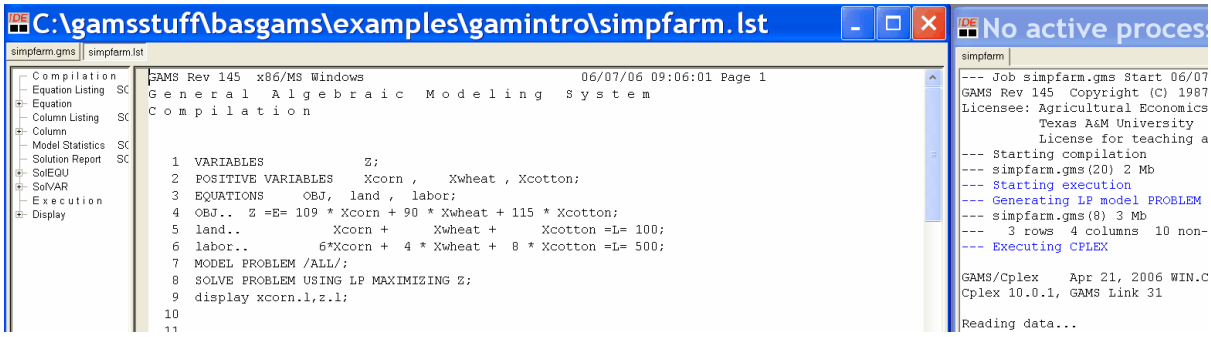


The positioning of your access is determined by the color of the line you double click on

© 2022 Prof. Bruce McCarl

Color of Line in Process Window	Function and Destination When Double Clicked.
Blue line	Jumps to the line in LST file corresponding to the blue line in process window. Blue lines also open put and other files created by the run.
Non-bolded black line	Jumps to location of last previous Blue Line in the LST file.
Red line	Identifies errors in source file. When you click on a red line the cursor jumps into the source (GMS) file at the location of the code that caused the error. Error description text appears in the process window and in the LST file that is not automatically addressed. When the shift key is held down you will go to the corresponding spot in the LST file.

We can navigate as we would with an editor or word processor, as we are automatically in the LST file (when clicking on a blue or black line) or GMS file when clicking on a red line in the IDE text editor.

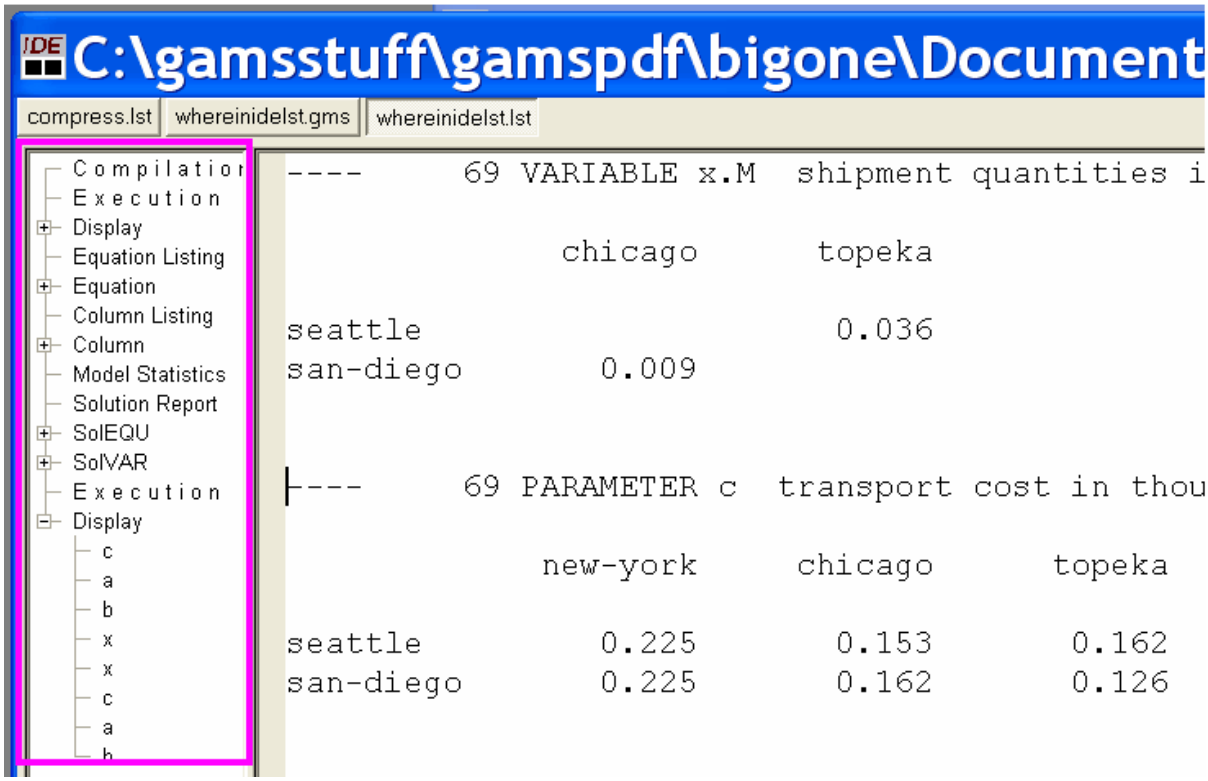


The file is frequently partially obscured by the process window. Is yours? You might want to narrow the process window to the side as in the picture above.

5.3.1.5.2 Using the LST file navigation w indow

By clicking on lines in the lst file navigation window you can access program output both in general and at particular locations.

© 2022 Prof. Bruce McCarl



The positioning of the cursor in the LST file is determined by the type of line you click on. A list of types of lines typically in the LST file is given below.

© 2022 Prof. Bruce McCarl

Name of Line in LST File Navigation Window	Function and Destination When Clicked.
Compilation	Jumps to top of echo print in LST file
Error Messages	Jumps to list of error messages when compilation messages incurred
Equation listing	Jumps to list of equation contents as illustrated.in the output section
Equation	Expandable allowing jump to beginning of list of contents for each individual equation block with contents as shown.in the output section
Variable listing	Jumps to list of variable contents as illustrated.in the output section
Variable	Expandable allowing jump to beginning of list of contents for each individual equation block with contents as shown.in the output section
Model statistics	Jumps to model statistics part of LST file as illustrated.in the output section
Solution Report	Jumps to model summary solution report
SolEQU	Expandable allowing jump to beginning of list of solution for each individual equation block with contents as shown.in the output section
SolVAR	Expandable allowing jump to beginning of list of solution for each individual variable block with contents as shown.in the output section
Execution	Jumps to beginning of post solve execution
Display	Expandable allowing jump to displays of specific parameters and other items

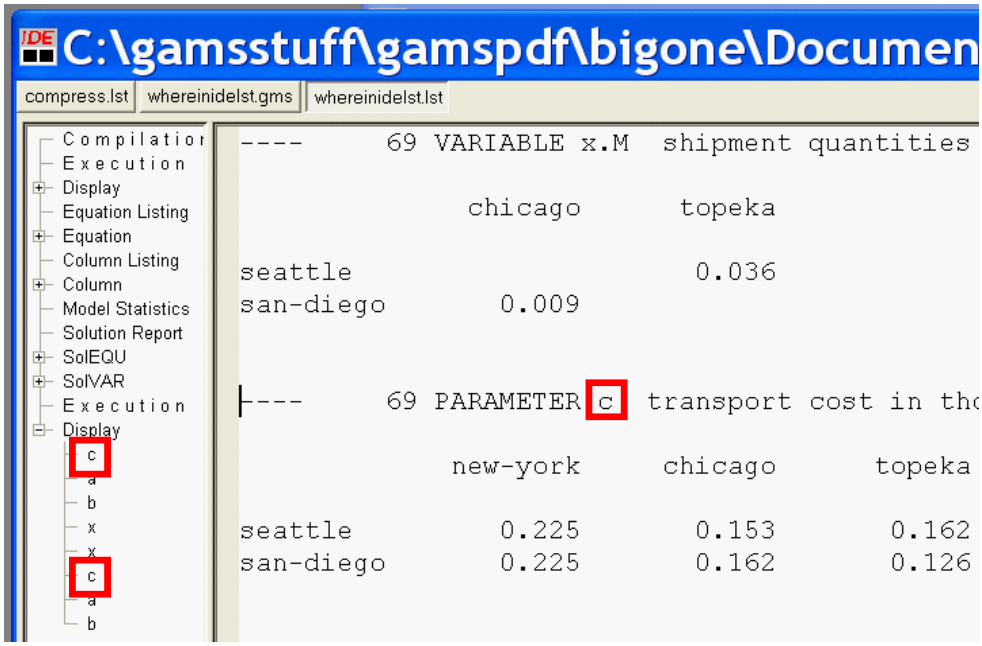
The width of the LST file navigation window is controlled by the user and can be narrowed as it has been above.

The data for this is stored in a LXI file that will be resident in your project directory with the same root as the associated LST file.

© 2022 Prof. Bruce McCarl

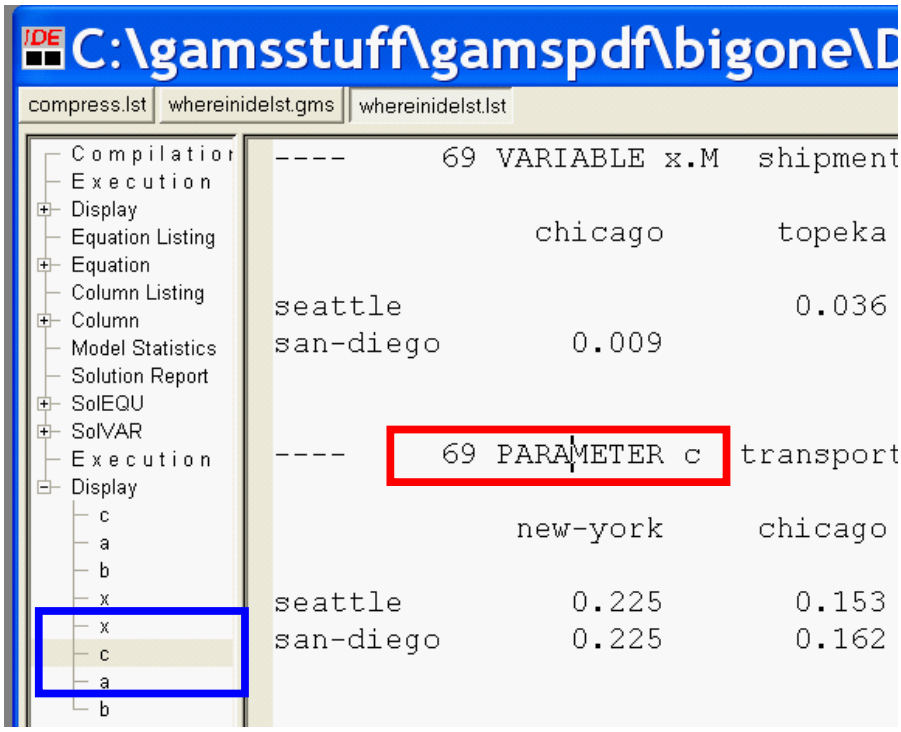
5.3.1.5.2.1 Finding the Active Location

When one is working in the LST file it is sometimes desirable to figure out where one is relative to the LST file navigation window. One does this by pressing control-mouse click on a LST file location. In particular suppose in a model an item is displayed more than once as in the case of the parameter **c** in the model whereinidelst.gms where one wishes to know which display statement is the one being viewed



Then if one presses control-click when the mouse is located at the point for which the location is desired then the IDE provides a gray background shade aound the associated item in the LST file navigation window.

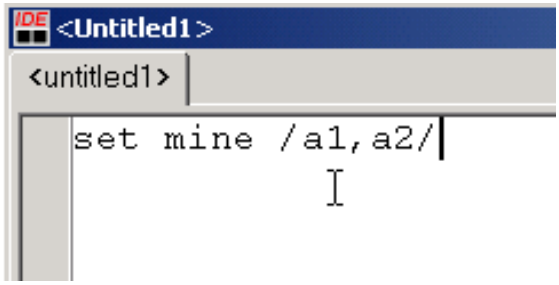
© 2022 Prof. Bruce McCarl



5.3.2 Working with your own file

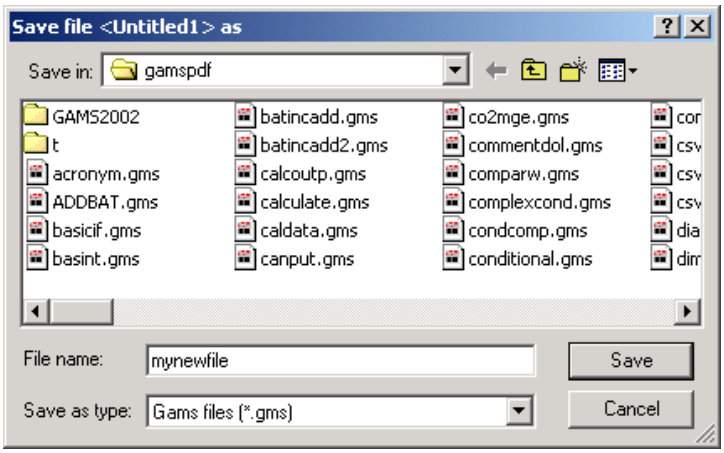
Now you are ready to work with your own files. You may already have a file or you may need to create one. There are two principal ways to do this:

- Open an existing GMS file. Then with the file menu save as dialogue change it's name. Now modify the contents to what you want. You may cut and paste as in other Windows programs.
- Open the file menu and use the new option. You will then get a file called noname which you may type GAMS instructions into



when you are finished save that file with whatever name you want. Note by default it will be assigned the extension .GMS.

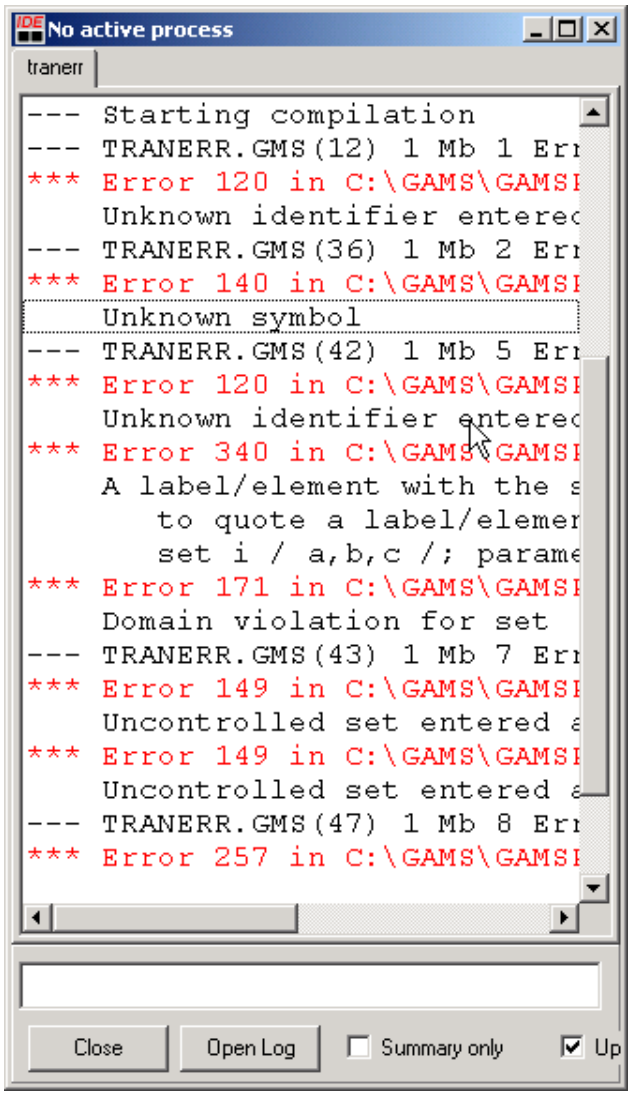
© 2022 Prof. Bruce McCarl



5.3.3 Fixing compilation errors

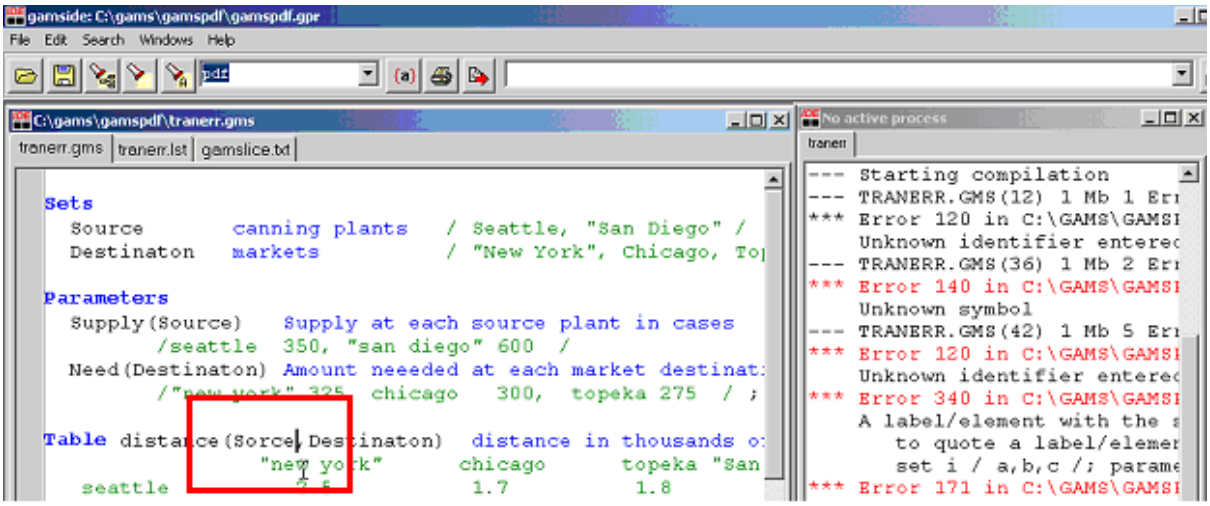
(tranerr.gms)

No one is perfect, errors occur in everyone's GAMS coding. The IDE can help you in finding and fixing those errors. Let's use the example **tranerr.gms** to illustrate how this occurs. A run of it yields the process window below



wherein the red lines mark errors. To see where the errors occurred lets double click on the top one. A double-click takes you to the place in the source where the error was made. The tip here is always start at the top of the process window when doing this so you find the first error as explained in the error proliferation section of the Compilation Errors chapter.

© 2022 Prof. Bruce McCarl



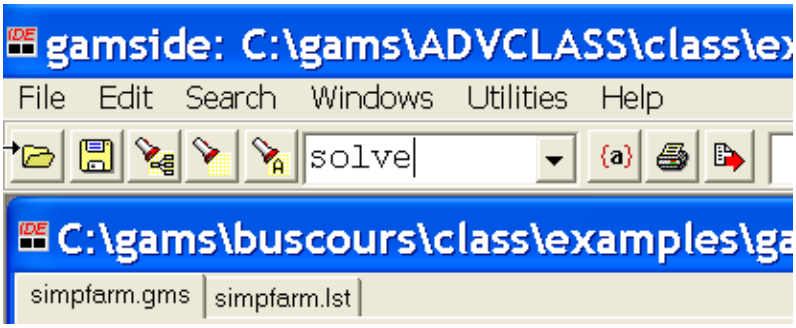
In this case the error is the spelling of source as sorce and note the IDE cursor as represented by the vertical line in the red box above is placed just at that spot.

5.3.4 Selected techniques for use of the IDE

Now suppose we cover a few powerful but sometimes overlooked aspects of the IDE.

5.3.4.1 Ways to find and/or replace text strings

The dialogs for finding text within the IDE involve use of the flashlight and the search window or the typing of the Keyboard command control F. The flashlight and search window involves the three icons with a flashlight in them and the box just to the right (that has the entry pdf in this case).



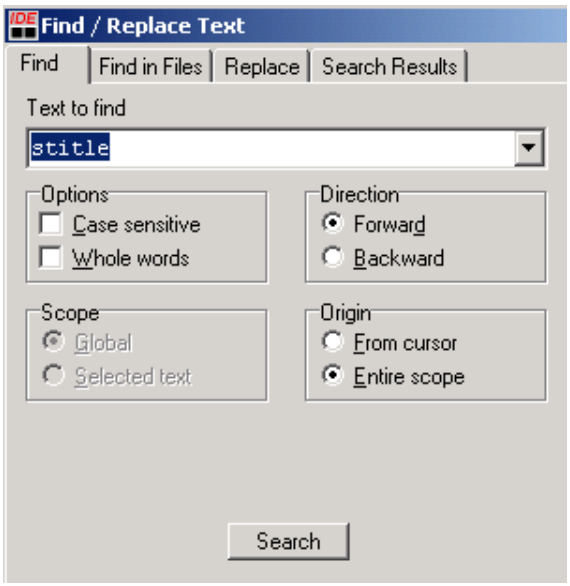
To find text you type the text string you are after in the search widow. In turn, clicking on the icon finds the first occurrence of what you want in the current file, while clicking on the icon finds the next occurrence in the current file. Finally, clicking on the icon finds all occurrences

in a specified group of files as discussed below.

You can also access search and replace through the search menu or by typing the keyboard shortcuts control f or control r. That dialogue allows access to more options and will search for the text under the current placement of the cursor.

5.3.4.1.1 Search menu and find in files

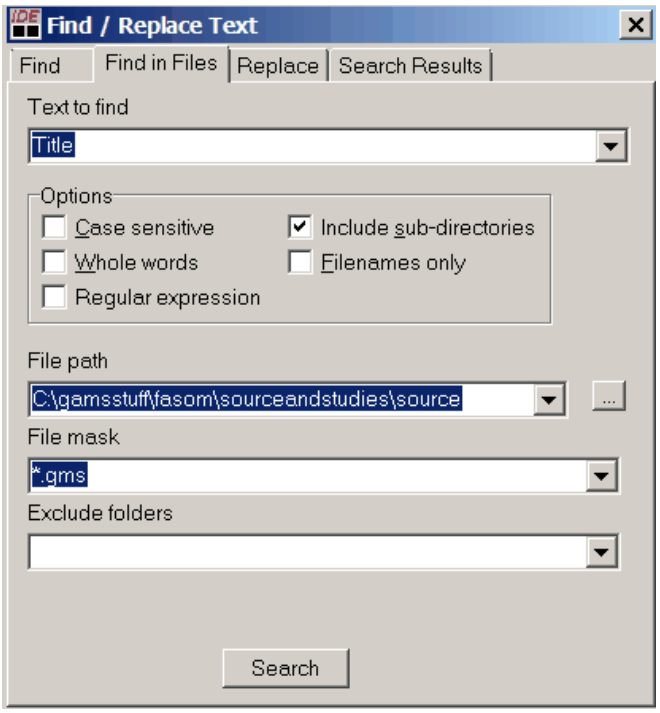
The search menu contains a number of options including IDE contains a useful find in files option. When you open this dialogue with control f you a window opens as follows



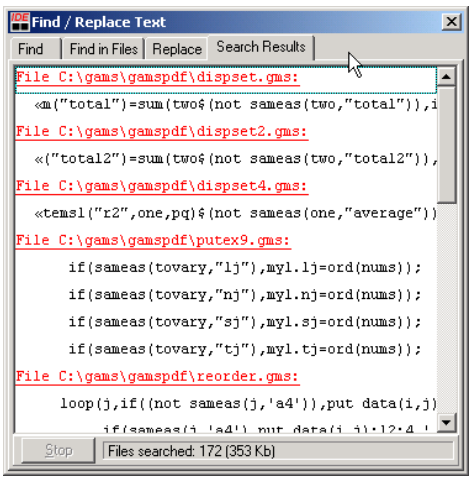
In that window the box gives the text to search for and one may use the boxes and buttons to influence the search direction, case sensitivity, scope etc.

When you open the find in files tab you see

© 2022 Prof. Bruce McCarl



In using this, a double click on the box with three dots to the right of File path lets you browse for a path and the file mask lets you choose the file types to search. You can also exclude some files from the search using the lowest box. You then search for the text in the text to find box. After a search you get a box as follows.



You may now navigate using this box. Clicking on a red line opens the file identified. Clicking on a black line opens the file and moves the cursor to the particular line.

5.3.4.2 Matching parentheses

(tranport.gms)

The IDE provides a way of checking on how the parentheses match up in GAMS code. This feature will

© 2022 Prof. Bruce McCarl

also match up { } or [ ]. This involves usage of the button containing the symbol from the menu bar or typing the F8 key coupled with appropriate cursor positioning. In particular positioning the cursor just after an open or close parenthesis and pressing the button or F8 jumps to the matching close or open parenthesis. For example, suppose we have a line of GAMS code like

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

and we position the cursor right after the first open parenthesis where the vertical line appears. Then clicking on the parenthesis matching button will jump the cursor to the position right after the matching ending parentheses as show below where the vertical line indicates the resultant cursor position. This will work whether it be 1, 100, or 1000+ lines away and vice versa. (Note it is not smart enough to ignore parentheses in comment statements so be sure any of those match up.) If a matching one is not found the cursor does not move.

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

Equivalently positioning the cursor right after an closing parenthesis and clicking on the button repositions the cursor right after the matching opening parenthesis. If a matching one is not found the cursor does not move.

5.3.4.3 Moving column blocks

The IDE allows one to move text conventionally through standard Windows copy, cut and paste operations. It also allows one to operate over column blocks of text. This again employs standard Windows copy, cut and paste operations. However in order to do this the column block must be designated. This is done by identifying the column block of text with the mouse or the keyboard by holding alt and shift down then moving the mouse or the cursor with the arrow keys.

```
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 turn copy, cut, and paste can be done with the Edit menu or with control c, x and v respectively as in normal windows. Control insert also pastes.

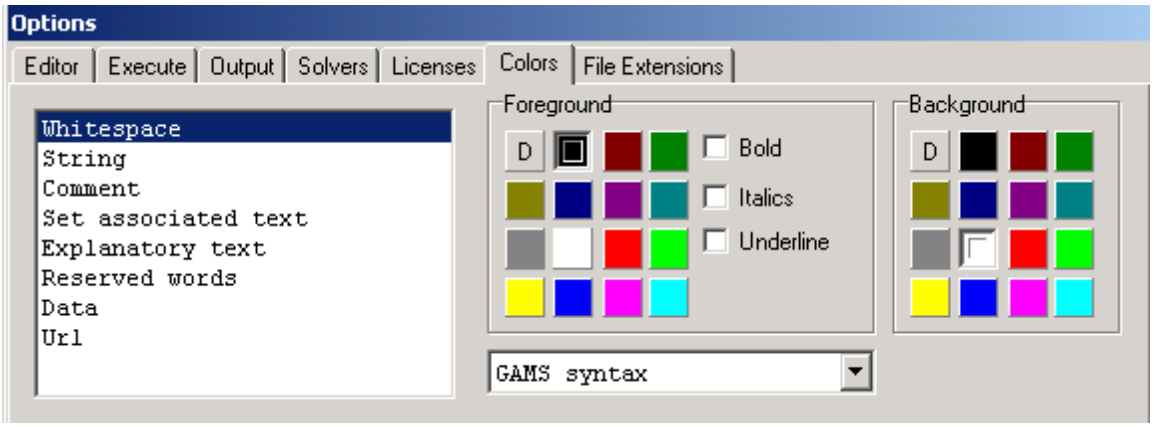
5.3.4.4 Altering syntax coloring

A feature in the IDE is GAMS related syntax coloring. The IDE recognizes a subset of the GAMS syntax and reflects this in the display colors. Note in the display below that commands, explanatory text and set elements are differentially colored.

© 2022 Prof. Bruce McCarl

```
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 ;
```

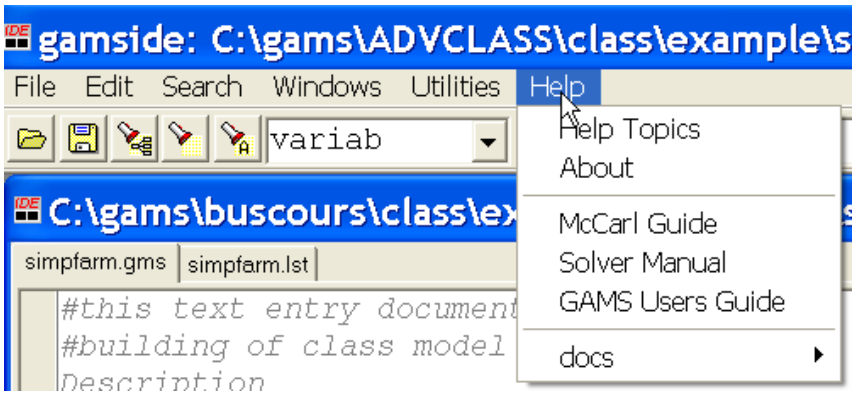
One can alter these syntax colors (as I have) through choices on the options menu under the colors tag



5.3.5 Finding out more through help

The IDE encompasses several paths to getting help on various GAMS related items and includes procedures for entering your own help content. When a user chooses Help the dialogue below appears

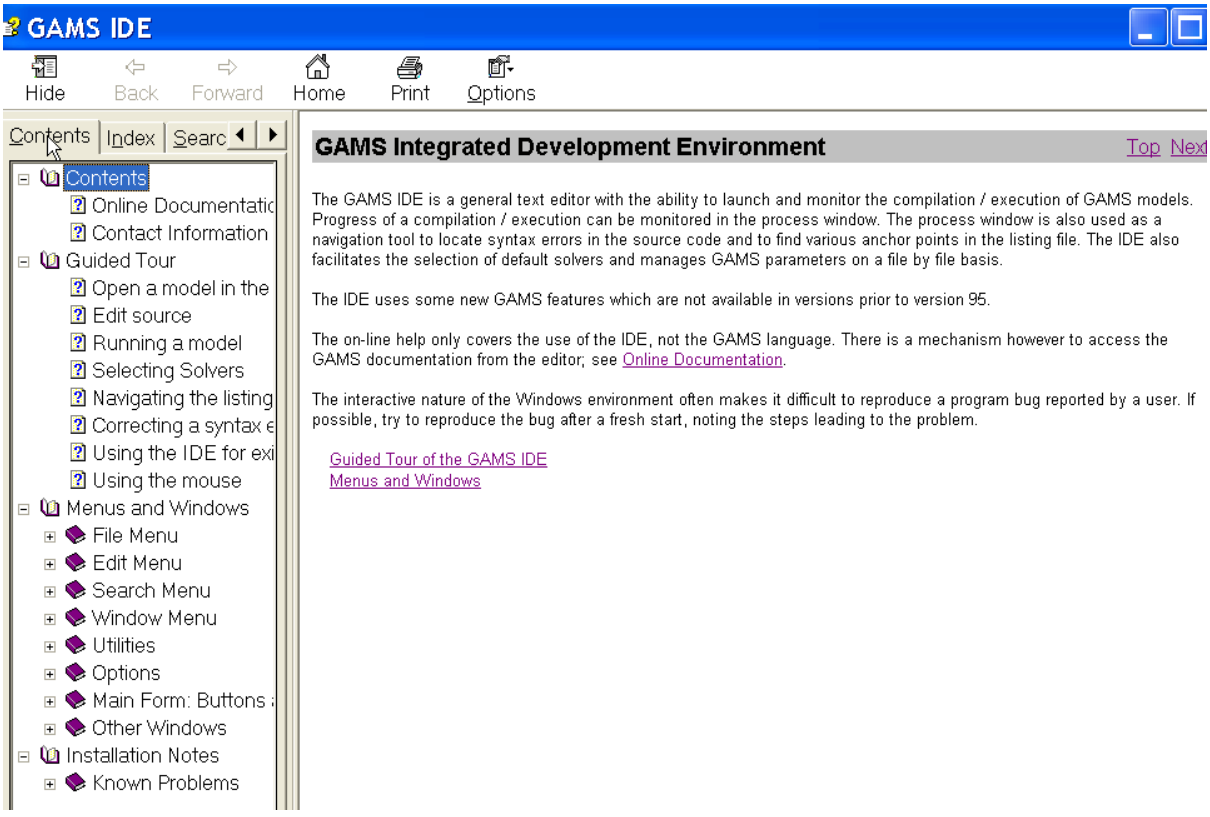
© 2022 Prof. Bruce McCarl



This allows access to help on several items.

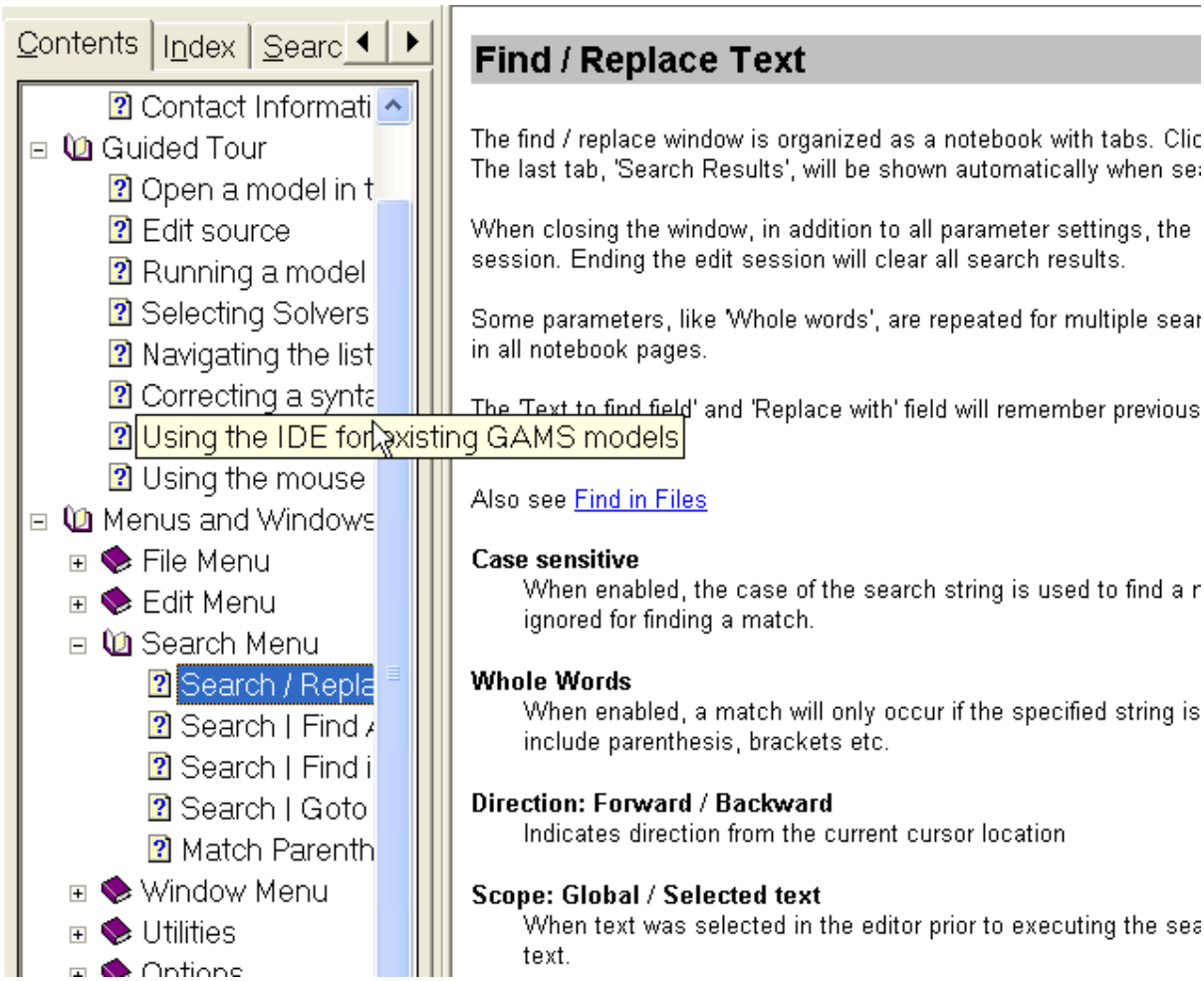
5.3.5.1 Help on the IDE

When a user chooses Help Topics one gets information on the IDE through the the dialogue that is captured below



that contains such things as

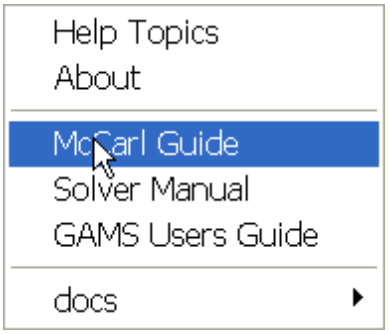
© 2022 Prof. Bruce McCarl



This and the other choices listed above are your real guide to the IDE, read it thoroughly.

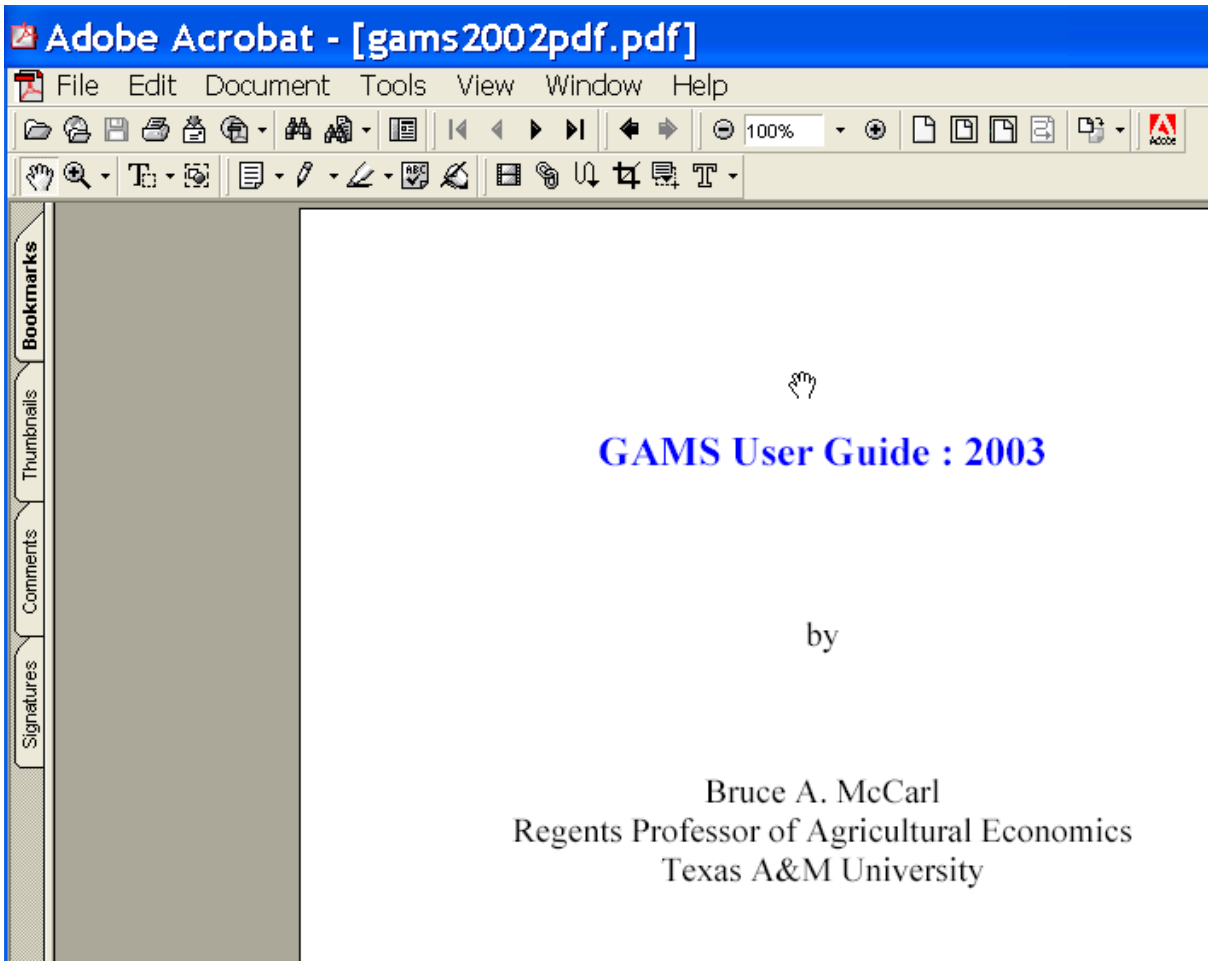
5.3.5.2 Help on GAMS

The IDE has a tie in to documentation. In particular suppose we wish to look at this reference guide on line. If we choose help and McCarl Guide

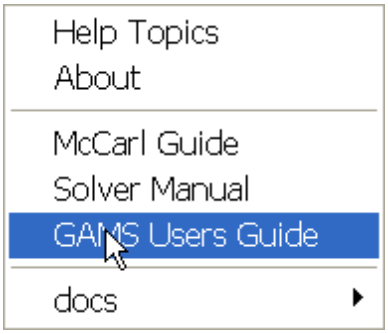


then provided Adobe Acrobat or another pdf reader is installed we get

© 2022 Prof. Bruce McCarl



while choice of



brings up the older User Guide

© 2022 Prof. Bruce McCarl

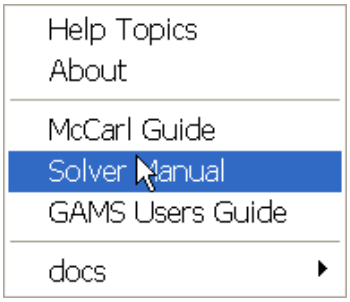
GAMS

A USER'S GUIDE

by:
Anthony Brooke
David Kendrick
Alexander Meeraus
Ramesh Raman

5.3.5.3 Accessing help on solvers

Using the Solver Manual choice allows us to get any of the available solver manuals



as shown below:

© 2022 Prof. Bruce McCarl

The Solver Manuals - Table of Contents

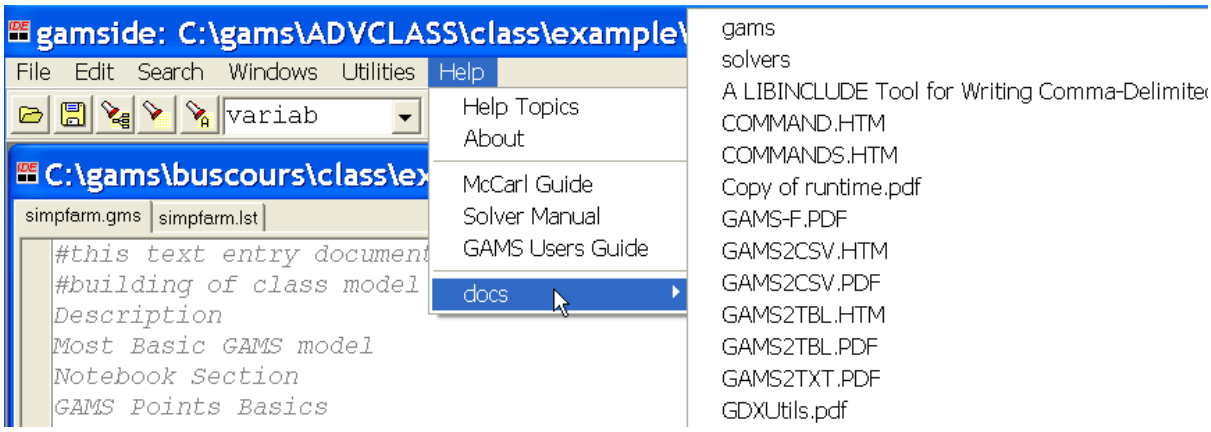
Solver Manuals	
Introduction	Using Solver Specific Options
BDMLP	LP solver that comes with any GAMS system
CONOPT	Large scale NLP solver from ARKI Consulting and Development
CPLEX	High-performance LP/MIP solver from Ilog
DECIS	Large scale stochastic programming solver from Stanford University
DICOPT	Framework for solving MINLP models. From Carnegie Mellon University
MILES	MCP solver from University of Colorado at Boulder that comes with any GAMS system
MINOS	NLP solver from Stanford University
MPSWRITE	MPS file generator that comes with any GAMS System
OQNLP	Multi-start method for global optimization from Optimal Methods Inc.
OSL	High performance LP/MIP solver from IBM
OSLSE	OSL Stochastic Extension for solving stochastic models
PATH	Large scale MCP solver from University of Wisconsin at Madison
SBB	Branch-and-Bound algorithm from ARKI for solving MINLP models
SCENRED	A tool for the reduction of scenarios modeling the random data processes
SNOPT	Large scale SQP based NLP solver from Stanford University
XA	Large scale LP/MIP system from Sunset Software
XPRESS	High performance LP/MIP solver from Dash
The complete Solvers Manual (Print Version)	
Other Solver Documents	
BARON	Branch-And-Reduce Optimization Navigator for proven global solutions from The Optimization Firm

which is a set of clickable links to pdf files documenting each of the available solvers.

5.3.5.4 Adding your own documentation

Users can augment the help menus by adding their own materials. In my classes I include a number of items under the choice mccarl and gamspdf we include the total mix of files composing this document. But users can add their own files (generally pdf or Html) that will appear on the help command (as in the mccarl entry above). Generally it is recommended that one put the additions in a subdirectory so the menu remains short. These are placed in the docs subdirectory under the GAMS system directory (Nominally in c:\program files\gams22.7\docs). In turn when one accesses help under the docs choice one gets a list of all the files placed therein.

© 2022 Prof. Bruce McCarl



5.3.5.5 Accessing documentation outside the IDE

Much of the documentation on GAMS and solvers discussed above is also accessible in other applications by navigating to the docs subdirectory of the GAMS system and opening the PDF files in the subdirectories thereof. The mccarl Guide is accesses through the /docs/userguides/mccarl subdirectory of the GAMS system directory (nominally c:\program files\gams22.7) and the file gams2002pdf.pdf or the file mccguide.html. Material on use of the guide is also in the file.

5.3.6 Unraveling complex files: Refreader

GAMS Modelers sometimes have to deal with complex implementations that

- Use include statements to incorporate numerous files.
- Have been developed by others.
- Have a complex structure with definitions and uses of items widely spread in a file or files.
- Contain items that are defined but never used.
- Were developed some time ago but are not extensively documented.

When faced with such cases one often asks

- Are there items defined in the program that are not used and if so what are they and where are they.
- Given an item in what files is it defined, declared and used in.

To resolve these questions a program called Refreader is included in the GAMSIDE.

5.3.6.1 Basic output

When Refreader runs it creates a window as follows

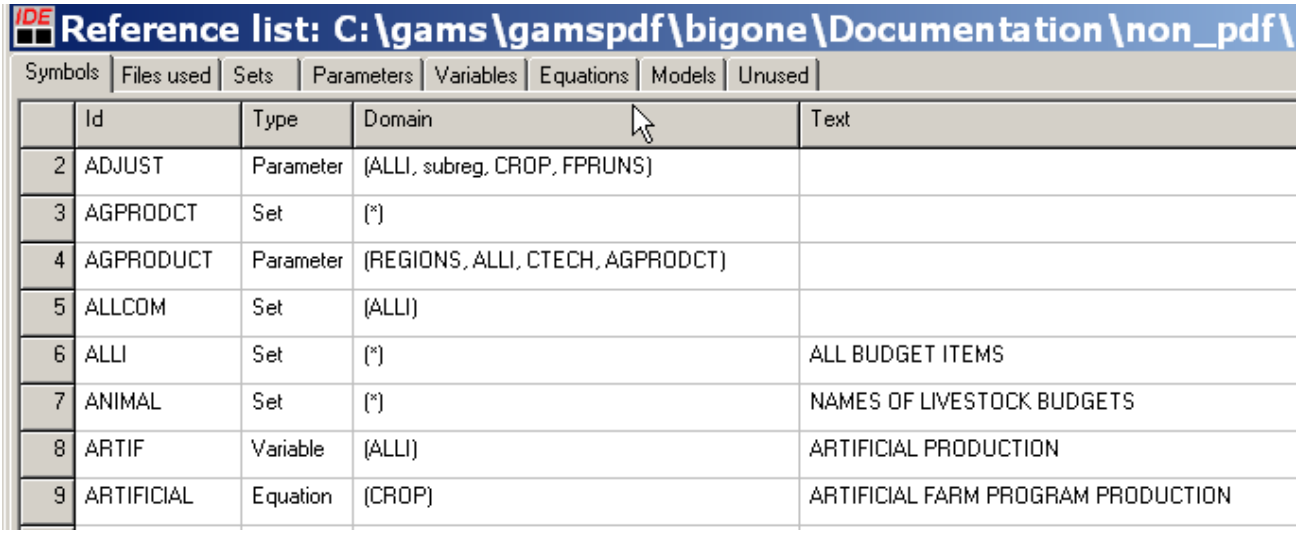
The tags in this window and their contents are

© 2022 Prof. Bruce McCarl

Symbol	a list of items that are declared or that are not used anywhere in the model
Files used	a list of the files included into the model
Sets	a list of the sets that exist in the program and the names of files in which they appear
Parameters	a list of the parameters (items defined in scalar, parameter or table statements) and the files in which they appear.
Variables	a list of the variables in the program and the names of files in which they appear
Equations	a list of the equations in the program and the names of files in which they appear
Models	a list of the models that exist in the program and the names of files in which they appear
Unused	a list of items that are declared or that are not used anywhere in the model

5.3.6.1.1 Symbol Tab

The symbol tab causes the output to appear as follows (for **asmall10.gms**)



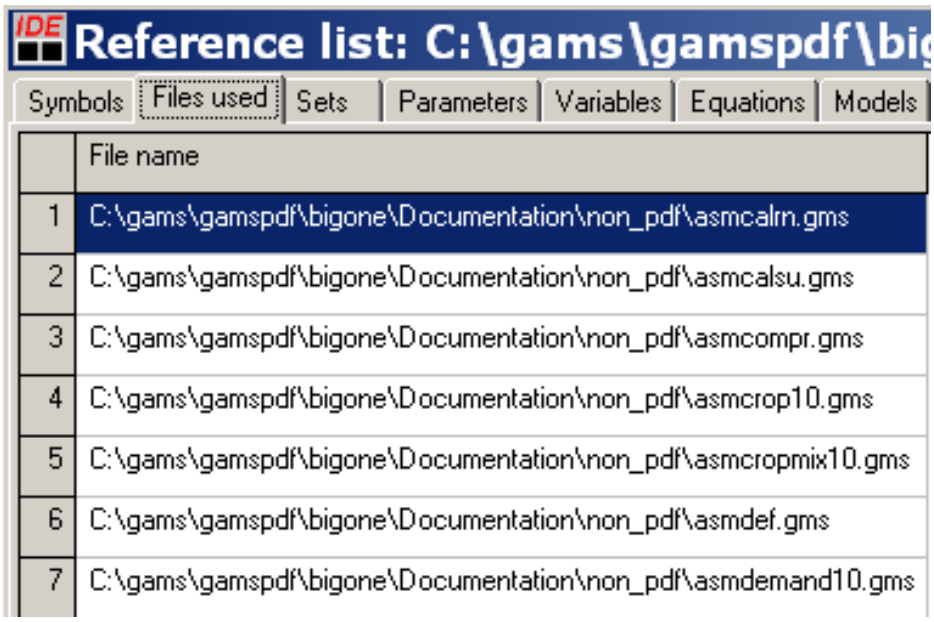
	Id	Type	Domain	Text
2	ADJUST	Parameter	(ALLI, subreg, CROP, FPRUNS)	
3	AGPRODCT	Set	(*)	
4	AGPRODUCT	Parameter	(REGIONS, ALLI, CTECH, AGPRODCT)	
5	ALLCOM	Set	(ALLI)	
6	ALLI	Set	(*)	ALL BUDGET ITEMS
7	ANIMAL	Set	(*)	NAMES OF LIVESTOCK BUDGETS
8	ARTIF	Variable	(ALLI)	ARTIFICIAL PRODUCTION
9	ARTIFICIAL	Equation	(CROP)	ARTIFICIAL FARM PROGRAM PRODUCTION

This shows all symbols in the program. The columns give the symbol name(ID), it's type(TYPE), the sets over which it is dimensioned (DOMAIN) and the explanatory text used in it's declaration (TEXT).

5.3.6.1.2 Files used Tab

The files used tab causes the output to appear as follows (again for the **asmall10.gms** example)

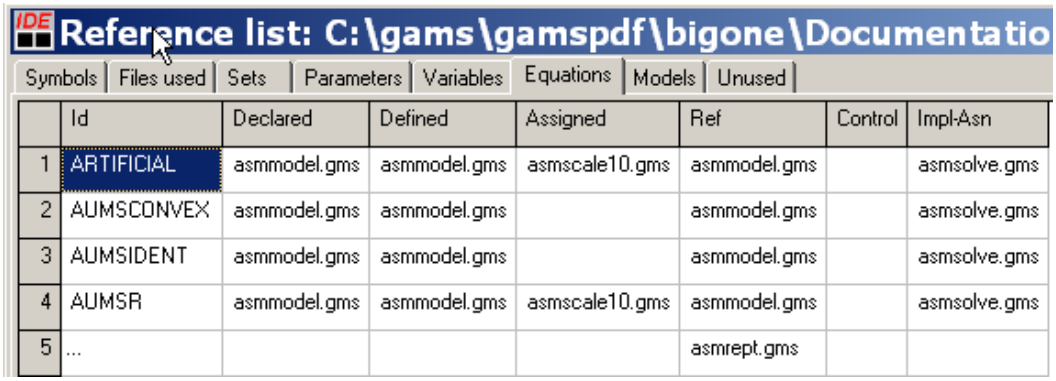
© 2022 Prof. Bruce McCarl



This gives the names of the files included in the program with their full path references.

5.3.6.1.3 Sets, Parameters etc. Tabs

Refreader contains 5 tabs that give information for sets, parameters etc. This display lists all items falling in a class (for all things that are sets or parameters etc.) one gets output as follows (for the equations in this case)



In this output the entries tell the names of the files in which certain things happen relative to the identified items. The categories of things include where items are

Declared	places where the named item is declared in a Set, Parameter, Table, Scalar, Variable, Equation, Acronym, File or Model command. This will be the first appearance.
Defined	places set elements or data are explicitly entered. For equations this tells where the .. specification begins.
Assigned	places where items appear on left hand side of an assignment statement
Ref	places where item is on left hand side of assignment statement or in a model

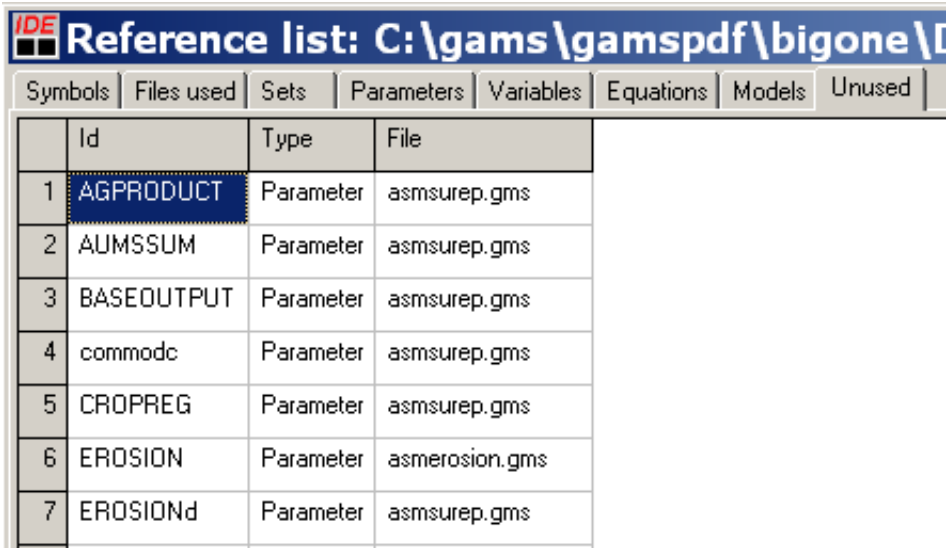
© 2022 Prof. Bruce McCarl

	equation
Control	places where set is used in controlling a sum or defining an equation
Impl-Assn	places where an equation or variable has data put into it by the results of a solve statement.

Double clicking on a file name opens that file within the IDE and generally indexes forward to the first reference of that type of the item named in the ID column in the file in that context (some exceptions occur when statements are spread over multiple lines particularly data definitions where the location of the / in parameter statements can cause problems).

5.3.6.1.4 Unused Tab

The unused tab identifies items that are declared (in set parameter etc statements) but are never used on the right hand side of an assignment (=) statement or in a model equation. The output for the model **asmall10.gms** is as follows



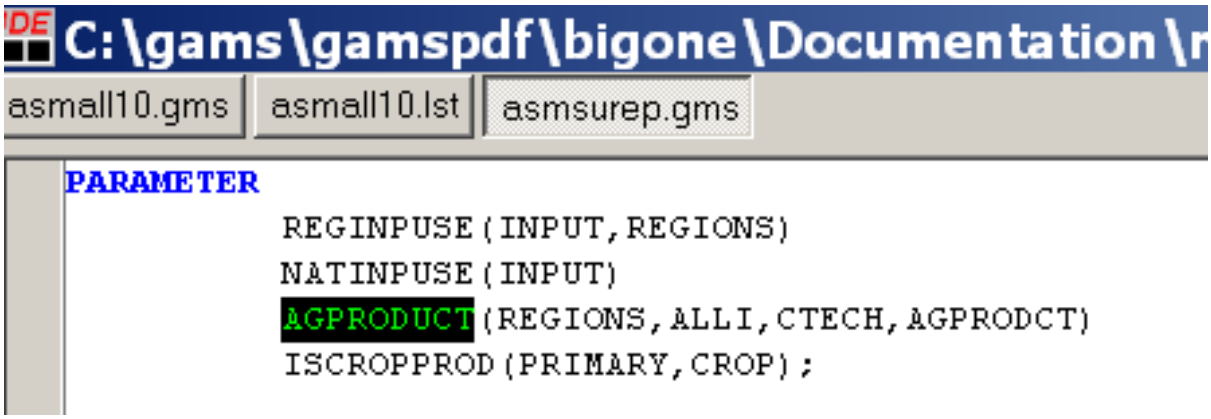
	Id	Type	File
1	AGPRODUCT	Parameter	asmsurep.gms
2	AUMSSUM	Parameter	asmsurep.gms
3	BASEOUTPUT	Parameter	asmsurep.gms
4	commodc	Parameter	asmsurep.gms
5	CROPREG	Parameter	asmsurep.gms
6	EROSION	Parameter	asmerosion.gms
7	EROSIONd	Parameter	asmsurep.gms

which shows

ID	names of items that are declared but unused
Type	item type (Parameter, set, variable, etc)
File	Name of file in which the item is declared.

Double clicking in the file column causes the IDE to open (if the IDE is the registered item to open GAMS files) to the spot in the file where the an item with the spelling of the name in the ID column appears in the context (declared, defined, referenced etc.) which generally will be the place where the declaration appears. For example if we double click in the file column associated with the **AGPRODUCT** row the IDE opens as follows

© 2022 Prof. Bruce McCarl



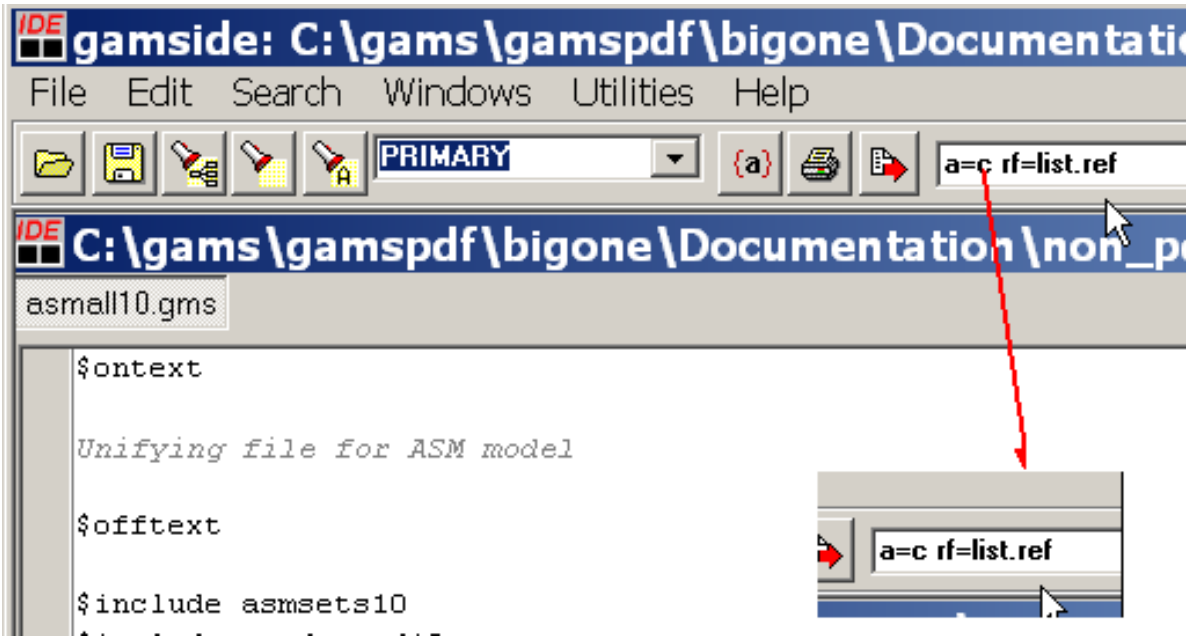
With **AGPRODUCT** highlighted.

5.3.6.2 Steps to Using Refreader

Refreader will only work after a particular "reference file" has been created by a GAMS run. The file is generated by adding the rf option to the command line call of GAMS. The general form of this is to either run

```
GAMS mymdel rf=filename.ref
```

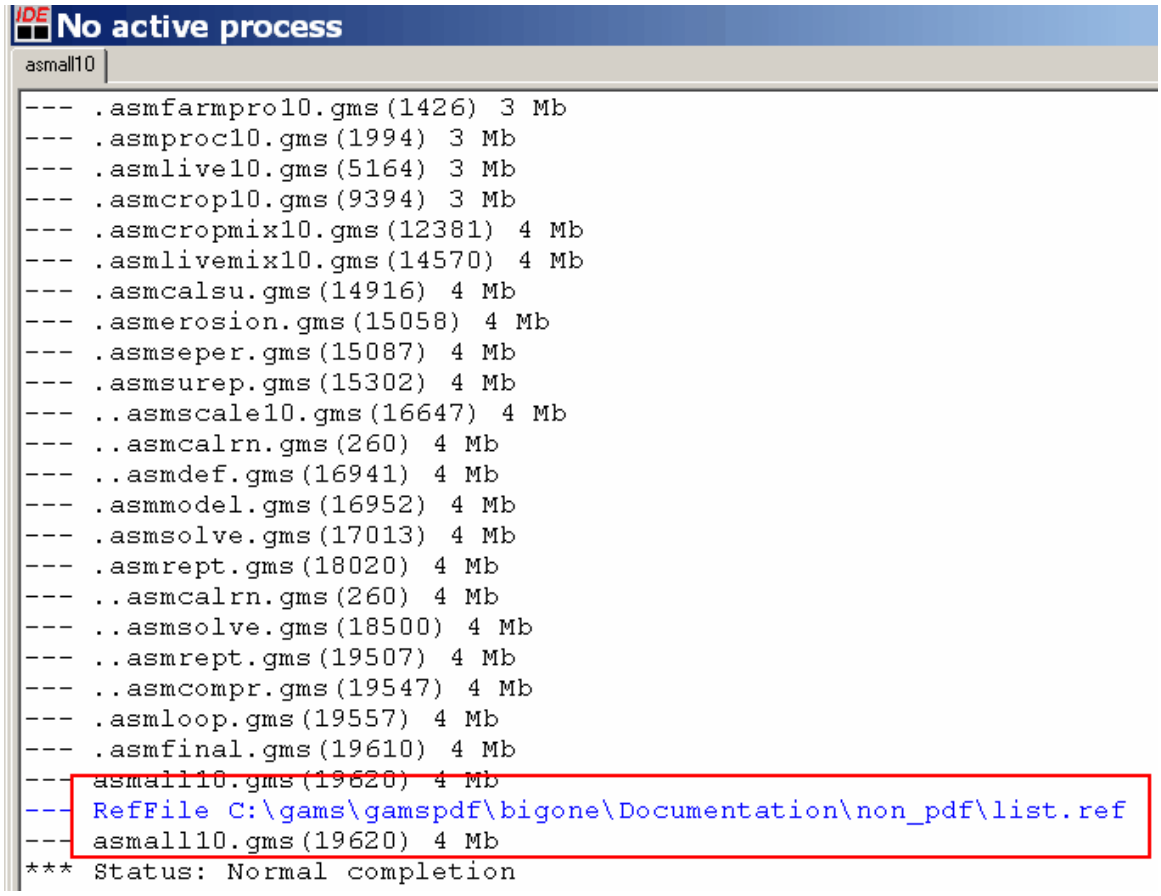
or enter a command in the command line box of the IDE as follows



The run of GAMS with the rf option places the name of the ref file in blue in the GAMS log file and double clicking on that line causes the refreader program to run and create its output. Several notes about the run

- The rf= command specifies the name of the file refreader will use. Generally refreader expects it to have the extension .ref. Typically we use list.ref.
- The rf file should be cleared out before the program is run as the rf command appends and does not overwrite.
- The rf file only covers the program components in a run and does not include any information from restart files. In general it is best to explicitly use all the files in one program without use of save and restart.
- It is often useful to just generate the reference file without any execution on behalf of the GAMS program. This is done by including the a=c option on the command line or in the command parameter IDE box.

Once the file has been run with the rf option the logfile is augmented with the blue line identifying the ref file name as below

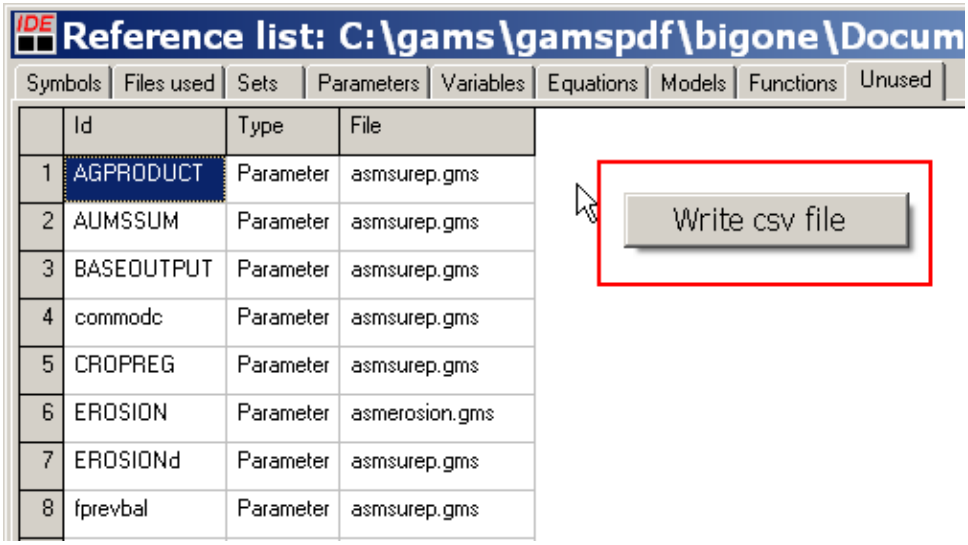


Double clicking on this opens the refreader window as discussed above.

5.3.6.3 Saving the Refreader output

The refreader program includes an option to save the output for inclusion in a program documentation or for other usages. This is done by right clicking in the reference file window whereupon a button that if pressed starts a write csv file dialogue as follows

© 2022 Prof. Bruce McCarl

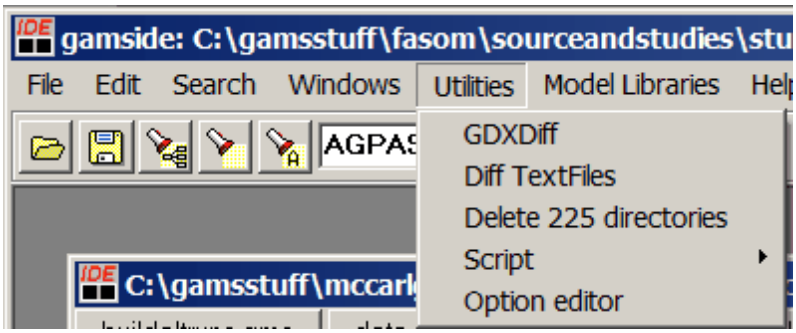


Clicking on that button will cause the refreader program to prompt for the name of a csv file in which the output will be saved. In turn, one may import that into other programs and eventually make tables for inclusion into application program documentations. Note if one reads the CSV into Excel and then pastes the tables in Excel into Word that Word tables will automatically be created.

5.3.7 Differencing files

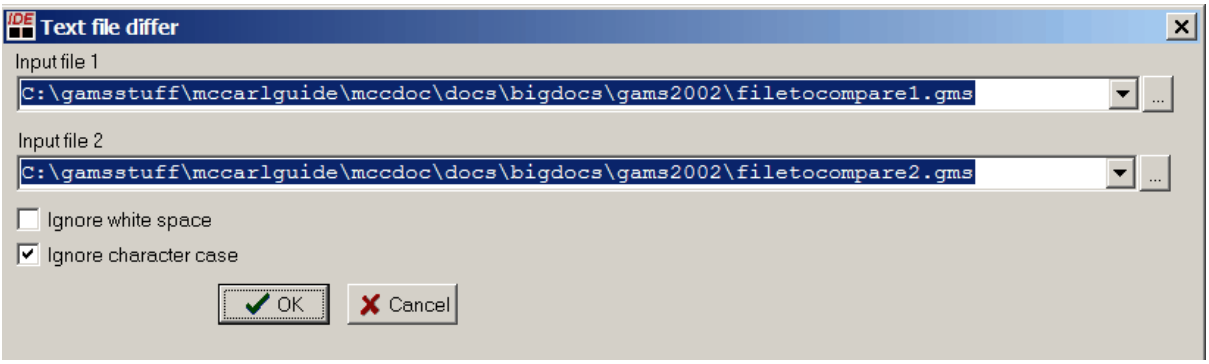
The IDE has now an integrated Text differencing feature that will compare two files in text or gms format and report on any found differences.

This is done through the GAMSIDE file menu using Utilities menu and the feature Diff Textfiles.



In turn one gets the window

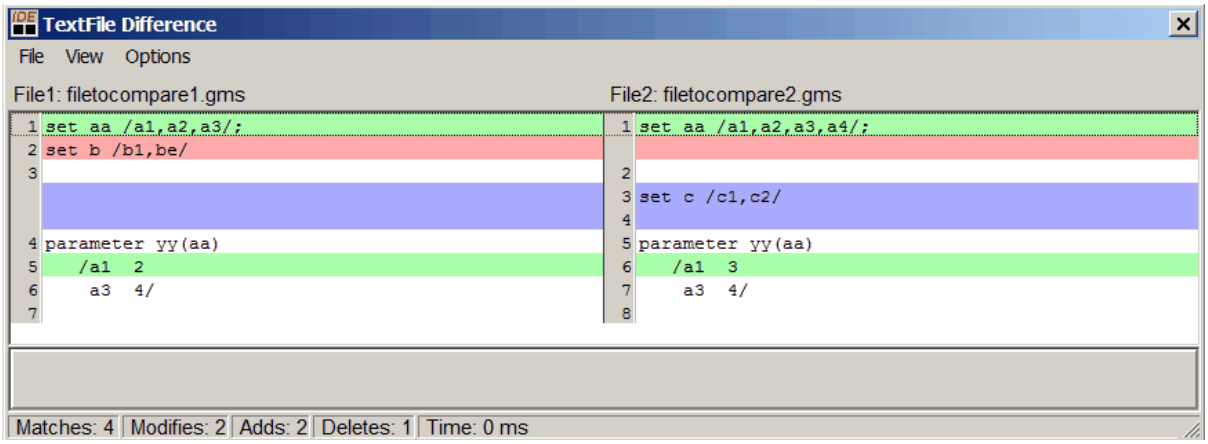
© 2022 Prof. Bruce McCarl



where one can specify the two files using the input file 1 and input file 2 boxes 1 and open 2 to choose the two files to compare.

Once that is done then selecting the OK button causes the files to be line by line compared.

The result for the comparison of example files filetocompare1.gms and filetocompare2.gms is given in the following window. When lines are identical in both files they appear with a white background. Green lines identify those with differences in similar lines. Reddish lines identify new lines in left hand file while blue identifies new content that appears in the right hand file.



In the near future this will move to the Utilities menu and will also remember the names of previous file comparisons (the release notes indicate this has happened but it has not in any release available as of this writing).

One can also doing this using the posix utility Diff as follows filecompare.gms

```
*get location of GAMS system directory
$setglobal root "%gams.sdir%"
*figure location or Expanded GAMS user guide files
$setglobal mccarlguide "%root%docs\bigdocs\gams2002\"
```

© 2022 Prof. Bruce McCarl

```
*identify name of first file to compare
$setglobal f1 %mccarlguide%filetocompare1.gms
*identify name of second file to compare
$setglobal f2 %mccarlguide%filetocompare2.gms

*show the control variables so you can check names are
right
$show

*invoke the difference  note the root and gbin parts
tell where the diff file is
$call '"%root%gbin\diff.exe"  %f1% %f2%'
```

5.3.8 Spell checking in files

The IDE contains spell checking features. One accesses this either through the Edit window or by performing a right mouse click in a file window for files other than LST files. In turn one selects **spelling** then can choose

- **Configure** causing a configuration screen to appear where one chooses the dictionary language along with having access to settings regarding items that will be checked and the degree of automatic correction.
- **Check** causing the whole document to be spell checked.
- **Check comments** where only comments are spell checked.
- **Check strings** where only explanatory text is spell checked.

5.3.9 Saving and Using a Script

The IDE has a facility to allow one to run a set of pre specified commands though the Utilities>script menu choice, A script can be a set of pre specified commands stored in a text file and can be built either manually or by recording commands much as one can do with spreadsheet macros.

The most important commands that can be used in a script are as follows

Fileopen - Opens a GAMS job
Filerun - Runs a GAMS job
Filewait - Waits until a running GAMS job is complete

The general format of these commands is

Fileopen;filename;
Filerun;filename;command line parameters
Filewait;filename;

where

filename gives the name of the file to be run which is proceeded by %ProjDir% if it is in the project directory and otherwise needs the full path specified.

© 2022 Prof. Bruce McCarl

command line parameters gives any command line parameters that are to be associated with the GAMS job.

Example

One could manually prepare a script file (script.txt) as follows

```
fileopen;%ProjDir%trandata.gms;
filerun;%ProjDir%trandata.gms; s=r1
filewait;%ProjDir%trandata.gms;
fileopen;%ProjDir%trandata.gms;
filerun;%ProjDir%tranmodl.gms; r=r1 s=r2
filewait;%ProjDir%tranmodl.gms;
fileopen;%ProjDir%trandata.gms;
filerun;%ProjDir%tranrept.gms; r=r2
filewait;%ProjDir%tranrept.gms;
```

which implements the save restart example from above.

One cam also use the command

Message;text to put in message box;

that inserts a yes no message box to inform the user but note this requires a mouse click answer and stops the job until it gets that answer.

Example (script1.txt)

```
*GAMSIDE script V1
filerun;%ProjDir%trandata.gms; s=r1
filewait;%ProjDir%trandata.gms;
message;Data job done   ;
filerun;%ProjDir%tranmodl.gms; r=r1 s=r2
filewait;%ProjDir%tranmodl.gms;
message;Model job done ;
filerun;%ProjDir%tranrept.gms; r=r2
filewait;%ProjDir%tranrept.gms;
message;Whole job done   ;
```

Notes

- The script is run by using utilities>script>play and then giving the name of the script file and pressing open.
- The script file itself must be saved before use if it is any different for the copy on the disk
- When the script begins all the files but the script file itself if open are saved so the GMS files will be updated
- Lines beginning with an * in column1 are treated as comments and ignored.
- When the filerun is performed the model is run with the run history placed in the process window and the LST file is automatically opened afterwards.
- The scripting is tricky to use and not very well supported in error messages so it is best to record the steps using the utilities>scropt>record dialogue then going through the steps with the mouse to encompass all one wants to do followed by a utilities>script>stop recording. The script is then placed in a text file and can be edited if needed.
- A number of other scripting commands are allowed as in the table below

© 2022 Prof. Bruce McCarl

Command	*Function*
ViewClose;fileidentifier;	Closes a file
FileClose;fileidentifier;	Closes a file
FileSave;fileidentifier;	Saves a file
FileSaveAll;fileidentifier;	Saves all files
FileCompile;fileidentifier;Command line parameters	Compiles a GAMS job

5.3.10 When is it not worth using?

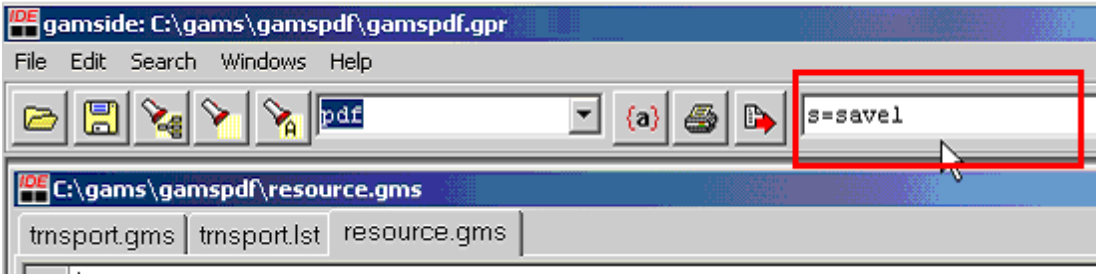
There are costs and benefits of these approaches. The IDE is much easier for simple models but is currently limited to PCs. The DOS/command line approach is generally better for models in customized environments. A development strategy for more complex implementations

- Use the IDE to get it right.
- Debug components of large models using save and restart.
- Then if more comfortable use DOS/UNIX with batch files such as

```
 GAMS mymodel -codex 1 -lo 0 -s ./t/save1
 call myprogram.exe
GAMS moremod  -lo 0 -r ./t/save1
```

5.3.11 Employing command line parameters

Experienced DOS or UNIX based GAMS users are used to having command line parameters associated with their GAMS execution commands. In the IDE a box is available just to the right of the execute button where we can associate a set of execution time parameters with a file. In turn note the IDE will remember these whenever the file is opened in this project in the future.



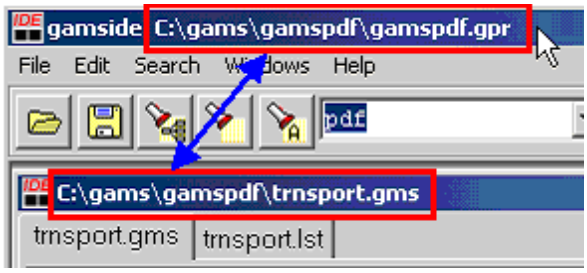
The IDE saves this file specific parameter information in the project file. This is particularly useful for save and restart parameters as once they are defined they are associated with every subsequent use of the file provided you're using the right project and have not changed the restart information.

© 2022 Prof. Bruce McCarl

5.3.12 A difficulty you will have using IDE

When using and teaching the IDE, I find that IDE project location in interaction with file placement gives almost everyone fits at some point or another. I have a rule of thumb to avoid problems.

- Make sure that you are working on files located in the same directory location as the project is located.
- This means making sure the path lines match up in the two dark blue locations in the screen shot below.



You do not have to follow this rule but deviations are the same as asking for trouble. When GAMS executes a file in a different directory it will look for options files, GCK files, include files etc in the directory where the project is located.

Another rule of thumb is also relevant.

- Whenever you need to work in a new directory define a new project.

5.3.13 Installation

Before beginning discussion of GAMS usage it is worthwhile to mention installation. Installation involves a number of tasks. There are

Install GAMS and on Windows machines the IDE
On Windows machines make IDE icon
Changing licenses
On Linux/Unix run Gamsinst
Choosing solvers
Unpacking software on Windows machines

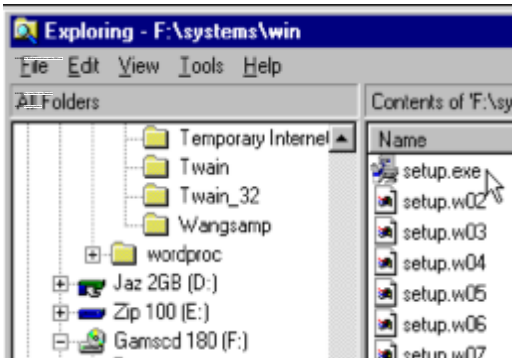
5.3.13.1 Install GAMS and on Windows machines the IDE

The IDE is automatically installed when GAMS is installed. To install both GAMS and the IDE on a windows machine (There are also more technical instructions in pdf files for **Windows** and **Unix/Linux**)

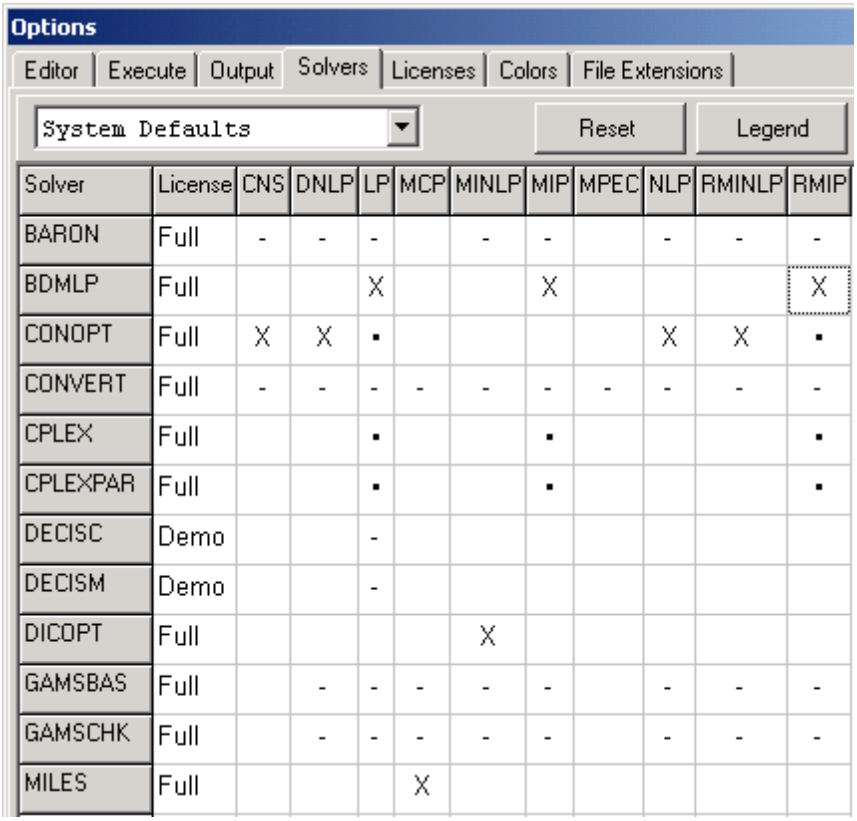
- Place the GAMS CD into your machine or download a new GAMS version that you have license rights to from the GAMS Corporation.
- Start the installation using the Windows Explorer or on non widows platforms find the lnxgams.sfx self extracting archive and run it. On windows machines go into the systems

© 2022 Prof. Bruce McCarl

subdirectory called **win** then double click on setup.exe



© 2022 Prof. Bruce McCarl



Solver	License	CNS	DNLP	LP	MCP	MINLP	MIP	MPEC	NLP	RMINLP	RMIP
BARON	Full	-	-	-		-	-		-	-	-
BDMLP	Full			X			X				X
CONOPT	Full	X	X	▪					X	X	▪
CONVERT	Full	-	-	-	-	-	-	-	-	-	-
CPLEX	Full			▪			▪				▪
CPLEXPAR	Full			▪			▪				▪
DECISC	Demo			-							
DECISM	Demo			-							
DICOPT	Full					X					
GAMSBAS	Full		-	-	-	-	-		-	-	-
GAMSCHK	Full		-	-	-	-	-		-	-	-
MILES	Full				X						

5.3.13.4.1 Solver choice outside of IDE

To revise solver choice on a non-windows machine you can

- Run Gamsinst from the GAMS system directory which will prompt for choice
- Employ the customization features as discussed in the options as discussed in Customizing GAMS chapter
- Edit (carefully) the file **GmscmpXX.txt** (where the XX depends on operating system Unix/ Linux -gmscmpun.txt; Windows 95/98 Gmscmp95.txt; Windows NT Gmscmpnt.txt) from the GAMS system directory changing the lines at the bottom specifying solver choice which appear just below

```
DEFAULTS
LP BDMLP
MIP BDMLP
RMIP BDMLP
NLP CONOPT
MCP MILES
CNS CONOPT
DNLP CONOPT
RMINLP CONOPT
MINLP DICOPT
```

© 2022 Prof. Bruce McCarl

5.3.13.5 Unpacking software on Windows machines

Sometimes one may obtain new software from GAMS off sites like *Rutherford's* in packed or zipped format. One can unpack this in the IDE using the update button that appears on the file options dialogue under the execute tab just to the right of the executable name box. This will allow unpacking of all zip or pck files in the GAMS system directory, but the user is prompted with a list before this happens.

6 Fixing Compilation Errors

The execution of a GAMS program passes through a number of stages, the first of which is the compilation step. Users watching the execution of a program are sometimes dismayed to get the message: **COMPILATION ERRORS** with the message indicating some large number of errors. These notes cover the process of finding and fixing GAMS compilation errors.

Don't bark up the wrong tree
Finding errors: ****
Finding errors: $
Repositioning error messages: Errmsg
Improperly placed semi colons - error A
Error message proliferation
Commonly found errors and their cause
Other common errors

6.1 Don't bark up the wrong tree

Before beginning a discussion of compilation error repair one thing needs mention. GAMS frequently marks compilation problems in latter parts of the code that are not really errors, but rather the messages are caused by errors in the earlier code. Case may occur where an omitted or extra semicolon or parenthesis in otherwise perfectly coded GAMS programs have caused hundreds of error messages. One should start fixing errors from the top and after fixing several errors rather than puzzling over obscure and often improper messages, rerun the compilation to find out if those repairs took to care of later marked errors. It is hardly ever desirable to try to fix all errors pointed out in one pass.

6.2 Finding errors: ****

When the screen or LOG shows compilation errors are present users should edit the .LST file directly or be guided through it by the IDE and look for the cause of the errors. Errors are marked by lines, which begin with 4 asterisks (****).

For example **errsemic.gms** one may find lines in the .LST file like the following

```
     3  SET PERIODS    TIME PERIODS          /T1*T5/;
     4      ELAPSED    ELAPSED TIME          /1*12/ ;
****                $140       $36
     5      PRODUCTS   LIST OF PRODUCTS      /WHEAT,STRAWBERRY/;
****                 $140   $36
```

which indicates errors were found in the **4th and 5th** lines of the input file.

6.3 Finding errors: $

The **** GAMS compilation error line contains information about the nature of the error. Error messages are numbered and placed below the place in the line they were encountered and begin with a $. In the example above, error number **140** occurred in line **4** and was caused by GAMS finding the word ELAPSED when it was looking for an instruction. In addition, a number **36** error was caused by the second incidence of the word ELAPSED and these errors were generated again by the same problem in line **5**. GAMS also includes a list of the error message numbers encountered and a brief description of the error at the bottom of the .LST file. In the case above the following appears at the bottom of the LST file:

Error Messages

```
 36    '=' or '..' operator expected - rest of statement ignored
108   Identifier too long
140   Unknown symbol .
```

Messages also appear in the LOG file and in the IDE the content of the LOG file is used as a navigation aid. Also in the IDE procedures are used to reveal exactly where in the source file the errors arise along with the offering up of the source file indexed to that position and ready for editing.

6.4 Repositioning error messages: Errmsg

It is possible to reposition where the error explanation appears. In particular, the location can be altered so the error message explanations appear just below the place the error is found mixed in with the source listing. This is done by using the option errmsg=1 in the GAMS command line. This can be imposed one of three ways.

- One can call GAMS with the command line parameter

```
gams mymodel errmsg=1
```

- When using the IDE this is placed in the GAMS command box in the upper right hand corner or if wanted for all models in the file option choice under the execute tab in the box for GAMS parameters.
- One can change the system level defaults by following the customization procedures entering this line in the file **gmcprm95.txt** on basic windows machines, which is also called **gmsprmnt.txt** on NT machines and **gmsprmun.txt** on Unix and Linux machines. The resultant file looks something like

```
*******************************************************
* GAMS 2.50 Default Parameterfile for Windows NT                *
* Gams Development Corp.                                        *
* Date : 20 Mar, 1998                                           *
```

© 2022 Prof. Bruce McCarl

```
*****************************************************
* entries required by CMEX, put in by gams.exe:
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
errmsg=1
ps=9999
optfile=1
```

In turn the output looks like the following

```
  6  SET PROCES   PRODUCTION PROCESSES /makechair,maketable,makelamp/
  7      RESOURCE  TYPES OF RESOURCES   /plantcap,salecontrct/;
  8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
****                          $120
**** 120  Unknown identifier entered as set
  9                /makechair 6.5 ,maketable 3, makelamp 0.5/
 10           PRODCOST(PROCESS)     COST BY PROCESS
 11                /Makechair 10  ,Maketable 6, Makelamp 1/
****                           $361
**** 361  Values for domain 1 are unknown - no checking possible
 12           RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
 13                /plantcap 10 ,salecontrct 3/;
 14  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
****                                              $362
**** 362  Values for domain 2 are unknown - no checking possible
 15                      Makechair   Maketable  Makelamp
 16       plantcap          3           2          1.1
 17       salecontrct       1          -1;
 18  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
 19  VARIABLES          PROFIT              TOTALPROFIT;
 20  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
 21                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE;
 22  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****                                                  $140
**** 140  Unknown symbol
```

6.5 Improperly placed semi colons - error A

What is wrong in the example above? The cause is what is probably the most common GAMS error for new users -- the placement of semi colons. GAMS commands should be terminated with a semi colon (**;**). However, commands can occupy more than one line. In the above case the original input looked like the following. (**errsemic.gms**)

```
SET PERIODS    TIME PERIODS          /T1*T5/;

    ELAPSED    ELAPSED TIME          /1*12/ ;
```

© 2022 Prof. Bruce McCarl

```
PRODUCTS    LIST OF PRODUCTS       /WHEAT,STRAWBERRY/;
```

This SET command is meant to continue for several lines, but the semi colon at the end of the first line terminates it. In turn, GAMS is looking for a command phrase in the second line and does not recognize the word ELAPSED, so it says **UNKNOWN SYMBOL**. The error is repeated at the end of the second line with yet another semi colon. There are two ways of fixing this. One may get rid of the semi colons in the first 2 lines only leaving a semi colon at the actual end of the SET declaration (i.e., the end of the third line) or one may enter the word SET on the second line and third lines.

Note GAMS does not strictly require a **semicolon** at the end of each command. In particular when the next line begins with one of the recognized GAMS keywords (SET, PARAMETER, EQUATIONS etc.) then a **semicolon** is assumed. However, it is good practice to terminate all commands with a **semicolon**. Certainly the lines before all calculations and equation specifications (.. lines) must have a **semicolon**.

6.6 Error message proliferation

The example also points out another common occurrence. GAMS usually generates multiple error messages as the consequence of a mistake. Once an error is encountered numerous messages may appear as the compiler disqualifies all further usages of the item in question and/or becomes confused. In the case above, subsequent references to the ELAPSED or PRODUCTS sets would cause errors and the SOLVE statement would be disqualified. Thus, users should fix the errors starting from the beginning and skip later errors if in doubt of their validity.

6.7 Commonly found errors and their cause

A number of errors are commonly found in GAMS that frequently confuse new users. Here I present a table of those errors with a brief indication of cause and a hyperlink cross-reference to longer sections that follow on common causes of such error messages. In using this table, readers should also look at the GAMS error message text as it may indicate additional causes.

GAMS Error Message	Potential Causes Discussion	Common Cause of Error
8	H	Mismatched parentheses-too many "(" found
36	I	Missing elements in equation definition
37	I	Missing equation type ("=L=", "=E=" "=G=") in equation specification
51-60	J	Illegal nonlinear specification
66	K	Item which has not been given numerical data appears in equation
71	I	Equation has been declared, but not algebraically specified with ".." statement

© 2022 Prof. Bruce McCarl

96	B	A statement ended and another began but no ; was included.
120	C, L	Cannot find a set with this name -- often a set element is referenced without properly being enclosed in "
125	F	Set is already in use in a sum or an equation definition
140	C, K , M	GAMS looking for a keyword or declared element and cannot find it. Check spelling and declarations.
141	K	Parameter without data used, or SOLVE does not proceed .L, and .M references
148	E	Item referenced with more or less indexed sets than in declaration
149	G, L	The set identified is not indexed either in a sum or an equation definition
170	C, D	Set element referred to cannot be found in set defined for this index position, check for misspelling, omissions, and references to wrong set
171	E, L, O	A domain error, Wrong set being referenced for this index position
195	N	Name used here duplicates that of an already defined item
198	P	Using ORD on a set that is not ordered
256	I, J, K	Something wrong with model specification. Look for other error messages immediately after solve statement
257	A-N	Solver not checked. Happens in conjunction with any GAMS error
340	L	Quotes likely forgotten around a specific set reference
408	H	Mismatched parentheses-too many ")" found

[1]**The entry below indicates when one gets error 8 a common cause of that error is discussed under the common error H section below.**

6.8 Other common errors

Many types of errors are possible in GAMS. I cannot cover each one. Thus, I list a set of common errors as well as an indication of what types of GAMS error messages they cause.

© 2022 Prof. Bruce McCarl

Excess or insufficient semi colons - error B
Spelling mistakes - error C
Omitted Set elements - error D
Indexing problems - error E
Summing over sets already indexed - error F
Neglecting to deal with sets - error G
Mismatched parentheses - error H
Improper equation ".." statements - error I
Entering improper nonlinear expressions - error J
Using undefined data - error K
Improper references to individual set elements - error L
No variable, parameter, or equation definition - error M
Duplicate names - error N
Referencing item with wrong set - error O

6.8.1 Excess or insufficient semi colons - error B

Too few or too many ;'s have been specified. The insufficient case is illustrated above. The example **shorterr.gms** provides a case where a semi colon has been omitted where the immediate first error message appears as below

```
  22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
  23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE
  24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****      $96                        $2   $195      $96
  25                        -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
****                        $409

Error Messages

96  Blank needed between identifier and text
       (-or- illegal character in identifier)
       (-or- check for missing ';' on previous line)
```

Normally this error is associated with GAMS error message $96.

6.8.2 Spelling mistakes - error C

Named sets, parameters, equations etc. may be referenced with a different spelling than in their declaration (i.e., the set CROPS is later referred to as CROP). GAMS identifies **set name misspellings with message $120**, **set element misspellings with $170** and **other misspellings with $140**. The example **shorterr05.gms** provides a case where the set PROCESS (**PROCES)**, one of its elements (**salecontrt)** and RESOURCEUSE (**RESOURUS)** are misspelled.

```
7  SET PROCEsS   PRODUCTION PROCESSES /makechair,maketable,makelamp/
8      RESOURCE  TYPES OF RESOURCES   /plantcap,salecontrct/;
9  PARAMETER PRICE(PROCES)      PRODUCT PRICES BY PROCESS
```

© 2022 Prof. Bruce McCarl

```
****                      $120
   10              /makechair 6.5 ,maketable 3, makelamp 0.5/
   11            Yield(process)  yields per unit of the process
   12                /Makechair 2   ,maketable 6 ,makelamp 3/
   13          PRODCOST(PROCESS)     COST BY PROCESS
   14                /Makechair 10  ,Maketable 6, Makelamp 1/
   15          RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
   16               /platcap 10 ,salecontrct 3/;
****                     $170
   17  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
   18                     Makechair   Maketable  Makelamp
   19       plantcap         3          2          1.1
   20       salecontrt       1         -1;
****                 $170
   21  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
   22  VARIABLES          PROFIT              TOTALPROFIT;
   23  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
   24                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE;
   25  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
   26                         -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
   27  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUS(RESOURCE,PROCESS)
****                                          $140

Error Messages

120  Unknown identifier entered as set
140  Unknown symbol
170  Domain violation for element
```

6.8.3 Omitted Set elements - error D

One can forget to include elements in set declarations. In turn, when these elements are referenced then an error arises (i.e., an error would occur if the element maketable was omitted from the declaration of set PROCESS but used when data were defined under the PARAMETER PRICE(PROCESS)) and subsequent sets. GAMS identifies such errors with message **$170**. The example **shorterr06.gms** provides a case where the element maketable is omitted from the set PROCESS.

```
    6  SET PROCESS   PRODUCTION PROCESSES /makechair,makelamp/
    7      RESOURCE  RESOURCES   /plantcap capacity ,salecontrct contract/;
    8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
    9                /makechair 6.5 ,maketable 3, makelamp 0.5/
****                                      $170
   10            Yield(PROCESS)  yields per unit of the process
   11                /Makechair 2   ,maketable 6 ,makelamp 3/
****                                      $170
   12          PRODCOST(PROCESS)     COST BY PROCESS
   13                /Makechair 10  ,maketable 6, Makelamp 1/
****                                      $170
   14          RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
   15               /plantcap 10 ,salecontrct 3/;
```

© 2022 Prof. Bruce McCarl

```
   16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
   17                      Makechair   Maketable  Makelamp
****                                          $170
   18       plantcap         3          2          1.1
   19       salecontrct      1         -1;

Error Messages
170  Domain violation for element
```

6.8.4 Indexing problems - error E

Parameters, variables, and equations are specified with a particular index order. Errors can be made where one inadvertently alters that order in subsequent references (i.e., RESOURUSE (RESOURCE,PROCESS) is referred to as RESOURUSE(PROCESS,RESOURCE)). One can also use **too many [(RESOURUSE(RESOURCE,PROCESS,resource)] or too few [RESOURUSE(RESOURCE)]** indices. Cases where the order of sets are changed are marked with message **$171**. Cases where **more** or less indices are used are marked with messages **$148**. The example **shorterr07.gms** provides cases, where permutations of the RESOURUSE(RESOURCE,PROCESS) are entered and the errmsg=1 option is used to reposition the messages.

```
  16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
  17                      Makechair   Maketable  Makelamp
  18       plantcap         3          2          1.1
  19       salecontrct      1         -1;
  20  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
  21  VARIABLES          PROFIT              TOTALPROFIT;
  22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
  23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE ;
  24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
  25                           -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
  26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(PROCESS,RESOURCE)
****                                                  $171     $171
**** 171  Domain violation for set
  27                       *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
  28  scalar x;
  29  x=sum((resource,process),RESOURUSE(RESOURCE,PROCESS,process));
****                                                        $148
**** 148  Dimension different - The symbol is referenced with more/less
****         indices as declared
30  x=sum(resource,RESOURUSE(RESOURCE));
****                                  $148
**** 148  Dimension different - The symbol is referenced with more/less
****         indices as declared
```

6.8.5 Summing over sets already indexed - error F

Errors occur when one treats the same SET more than once [(i.e., process is summed over twice in the expression

```
sum((resource,process),SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)));]
```

© 2022 Prof. Bruce McCarl

or where an equation is defined over a set and one tries to sum over it. For example in the following case RESOURCE defines the equation and is summed over

```
resource2(resource,process)=sum(process,RESOURUSE(RESOURCE,PROCESS));
```

Such errors are marked with message $125. The example **shorterr08.gms** illustrates such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

```
  26 AVAILABLE(RESOURCE).. SUM((PROCESS,RESOURCE),RESOURUSE(RESOURCE,PROCESS)
****                                              $125
**** 125  Set is under control already
  27                     *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
  28  scalar x;
  29  x=sum((resource,process),SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)));
****                                            $125
**** 125  Set is under control already
  30  parameter resource2(resource,process);
  31  resource2(resource,process)=sum(process,RESOURUSE(RESOURCE,PROCESS));
****                                            $125
**** 125  Set is under control already
```

6.8.6 Neglecting to deal with sets - error G

Errors occur when one does not sum or index over a set referenced within an equation (i.e., in the example **shorterr09.gms** the set **Process** is used occurs but is not summed over or used in defining the equations). This is marked with message $149.

```
  26  AVAILABLE(RESOURCE).. RESOURUSE(RESOURCE,PROCESS)
****                                                $149
**** 149  Uncontrolled set entered as constant
  27                       *PRODUCTION(PROCESS)  =L= RESORAVAIL(RESOURCE);
****                                             $149
**** 149  Uncontrolled set entered as constant
  28  scalar x;
  29  x=sum(resource,RESOURUSE(RESOURCE,PROCESS));
****                                              $149
**** 149  Uncontrolled set entered as constant
  30  parameter resource2(resource);
  31  resource2(resource)=RESOURUSE(RESOURCE,PROCESS);
****                                                 $149
**** 149  Uncontrolled set entered as constant
```

6.8.7 Mismatched parentheses - error H

Parentheses must match up in expressions. An excess number of open "(" parentheses are marked with **$8** while excess closed ")" parentheses are marked with **$408** [i.e., cases like SUM (I,X(I); or SUM (I,X(I))); generate errors] and but other errors can enter. The example **shorterr10.gms** illustrates such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

© 2022 Prof. Bruce McCarl

```
  OBJT.. PROFIT=E=   SUM(PROCESS,((PRICE(PROCESS)*yield(process)
  25                             -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
****                                                                    $8
****   8  ')' expected
  26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
  27                       *PRODUCTION(PROCESS)))  =L= RESORAVAIL(RESOURCE);
****                                          $37,408$409
****  37  '=l=' or '=e=' or '=g=' operator expected
**** 408  Too many ),] or }
**** 409  Unrecognizable item - skip to find a new statement
****        looking for a ';' or a key word to get started again
  28  scalar x;
  29  x=sum((resource,process),RESOURUSE(RESOURCE,PROCESS)));
****                                                      $408
**** 408  Too many ),] or }
  30  x=sum((resource,process),(RESOURUSE(RESOURCE,PROCESS));
****                                                       $8
****   8  ')' expected
```

Two error prevention strategies are possible when dealing with parentheses.

- Many editors, including the one in the IDE, contain a feature that allows one to ask the program to identify the matching parentheses with respect to the parenthesis that is sitting underneath the cursor. It is highly recommended that GAMS users employ this feature during model coding to make sure that parentheses are properly located for the end of sums, if statements, loops etc.
- Alternative characters can be used in place of parentheses. In particular, the symbols { } or [ ] can be used instead of the conventional ( ). GAMS is programmed to differentially recognize these symbols and generate compile errors if they do not match up. Thus a statement such as

```
x = sum( j, ABS ( TTS (j) ) );
```

  can be restated as

```
x = sum[ j, ABS { TTS(j) } ];
```

  Such a restatement would provide a visual basis for examining whether the parentheses were properly matched. It would also generate errors if one did not use the alternative parenthesis forms in the proper sequence. For example, the following statement would stimulate compiler errors:

```
x = sum[ j, ABS { TTS(j} ] );.
```

6.8.8 Improper equation ".." statements - error I

Each declared equation must be specified with a statement, which contains certain elements. Omitting the ".." causes error $36. Omitting the equation type ("=L=", "=E=", or "=G=") causes error $37.

© 2022 Prof. Bruce McCarl

Omitting the specification of a declared equation is marked with messages $71 and $256. The example **shorterr11.gms** illustrates the first two such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

```
  25  OBJT PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
****            $36
****  36  '=' or '..' or ':=' or '$=' operator expected
****      rest of statement ignored
  26                            -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
  27  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
  28                       *PRODUCTION(PROCESS))  = RESORAVAIL(RESOURCE);
****                                                                  $37
****  37  '=l=' or '=e=' or '=g=' operator expected
```

while the example **shorterr12.gms** illustrates the last two

```
  22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
  23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE
  24                     notthere one i forgot ;
  25  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
  26                            -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
  27  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
  28                       *PRODUCTION(PROCESS))  =l= RESORAVAIL(RESOURCE);
  29  MODEL RESALLOC /ALL/;
  30  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                          $71,256
****  71  The symbol shown has been declared as an equation, but no
****         Symbolic equation (..) was found. hint - look for commas in the
****         Documentation text for the equations. use quotes around the
****         Text or eliminate the commas.
**** 256  Error(s) in analyzing solve statement. More detail appears
****      Below the solve statement above
**** The following LP errors were detected in model RESALLOC:
**** 71 notthere is an undefined equation
```

6.8.9 Entering improper nonlinear expressions - error J

One gets messages **$51-$60 and $256** containing the word ENDOGENOUS when the equations contain nonlinear terms beyond the capability of the solver being used (i.e., nonlinear terms do not work in LP solvers). The example **shorterr13.gms** illustrates the first two such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

```
  26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
  27                      *sqr(PRODUCTION(PROCESS)))  =L= RESORAVAIL(RESOURCE);
  28  MODEL RESALLOC /ALL/;
  29  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                          $51,256
****  51  Endogenous function argument(s) not allowed in linear models
```

© 2022 Prof. Bruce McCarl

```
**** 256  Error(s) in analyzing solve statement. More detail appears
****      Below the solve statement above
**** The following LP errors were detected in model RESALLOC:
**** 51 in equation AVAILABLE .. VAR argument(s) in function
```

6.8.10 Using undefined data - error K

When data items are used which have **not been declared** (in a TABLE, PARAMETER or SCALE statement) **one gets told they are an unknown symbol via error $140** simply stating **GAMS doesn't know what they are**. In addition, when declared **items are used which have not received numerical values**, one gets either: 1**) message $141 when the items are used in calculations**, or 2**) messages $66 and $256 when the items are used in model equations**. One can also get message $141 when referring to optimal levels of variables (i.e., X.L or X.M) when a SOLVE has not been executed. The example **shorterr14.gms** illustrates the first two such errors under the use of the errmsg=1 option which repositions the error message explanatory text. This example also shows how **$141 errors occur for all uses of .L and .M commands when GAMS stops and the actions undertake by solve are not checked because of earlier compiler errors**.

```
  6  SET PROCESS   PRODUCTION PROCESSES /makechair,maketable,makelamp/
  7  RESOURCE  RESOURCES  /plantcap ,salecontrct ;
  8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
  9               /makechair 6.5 ,maketable 3, makelamp 0.5/
 10            Yield(process)  yields per unit of the process
 11            RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
 12               /plantcap 10 ,salecontrct 3/;
 13  parameter RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
 14  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
 15  VARIABLES          PROFIT              TOTALPROFIT;
 16  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
 17                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE ;
 18
 19  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
 20                             -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
****                                $140
**** 140  Unknown symbol
 21  scalar x;
 22  x=sum(PROCESS,Yield(process));
****                    $141
**** 141  Symbol neither initialized nor assigned
****         A wild shot: You may have spurious commas in the explanatory
****         text of a declaration. Check symbol reference list.
 23  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
 24                       *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
 25
 26  MODEL RESALLOC /ALL/;
 27  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                        $257
**** 257  Solve statement not checked because of previous errors
 30  solprod(PROCESS)= PRODUCTION.l(PROCESS);
****                               $141
```

© 2022 Prof. Bruce McCarl

```
**** 141  Symbol neither initialized nor assigned
****         A wild shot: You may have spurious commas in the explanatory
****         text of a declaration. Check symbol reference list.
```

while **shorterr15.gms** illustrates what happens when items without numerical value are used in equations.

```
  16  parameter RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE;
  17  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
  18  VARIABLES          PROFIT              TOTALPROFIT;
  19  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
  20                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE ;
  21
  22  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
  23                             -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
  24  scalar x;
  25  x=sum(PROCESS,Yield(process));
  26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
  27                       *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
  28
  29  MODEL RESALLOC /ALL/;
  30  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                          $66,256
****  66  The symbol shown has not been defined or assigned
****         A wild shot: You may have spurious commas in the explanatory
****         text of a declaration. Check symbol reference list.
**** 256  Error(s) in analyzing solve statement. More detail appears
****      Below the solve statement above
**** The following LP errors were detected in model RESALLOC:
**** 66 RESOURUSE has no data
```

and **shorterr15.gms** shows what happens when a solve statement is not present and .L or .M **variable or equation attributes** are used in calculations.

```
  29  MODEL RESALLOC /ALL/;
  32  solprod(PROCESS)= PRODUCTION.l(PROCESS);
****                                $141
**** 141  Symbol neither initialized nor assigned
****         A wild shot: You may have spurious commas in the explanatory
****         text of a declaration. Check symbol reference list.
```

6.8.11 Improper references to individual set elements - error L

Individual set elements are referenced by entering their name surrounded by quotes. When the **quotes are not entered one gets message $120** and **when the item is in the set relevant to this place without m quotes one gets $340** (i.e., if I have defined X(CROP) with CORN as an element in CROP, then X(CORN) is wrong, but X("CORN") is right). One also gets errors assuming this is a new **set indicating it has not been** dealt with **$149** and this is **not the set the item is defined over $171** (a domain error). The example **shorterr16.gms** illustrates such errors under the use of the errmsg=1

© 2022 Prof. Bruce McCarl

option which repositions the error message explanatory text.

```
  6  SET PROCESS   PRODUCTION PROCESSES /makechair,maketable,makelamp/
  7      RESOURCE  TYPES OF RESOURCES   /plantcap capacity ,salecontrct
                                                                  contract/
  8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
  9                /makechair 6.5 ,maketable 3, makelamp 0.5/
 10            Yield(process)  yields per unit of the process
 11                /Makechair 2   ,maketable 6 ,makelamp 3/
 12           PRODCOST(PROCESS)     COST BY PROCESS
 13                /Makechair 10  ,Maketable 6, Makelamp 1/
 14           RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
 15                /plantcap 10 ,salecontrct 3/;
 16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
 17                     Makechair   Maketable  Makelamp
 18       plantcap         3           2          1.1
 19       salecontrct      1          -1;
 20  scalar x;
 21  x=price(makechair);
****                  $120,340$149,171
**** 120  Unknown identifier entered as set
**** 149  Uncontrolled set entered as constant
**** 171  Domain violation for set
**** 340  A label/element with the same name exist. You may have forgotten
****        to quote a label/element reference. For example,
****        set i / a,b,c /; parameter x(i); x('a') = 10;
```

6.8.12 No variable, parameter, or equation definition - error M

When a variable, parameter, or equation is **used which has not been declared** one gets error **$140**. The example **shorterr17.gms** illustrates such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

```
 20  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
 21  VARIABLES          PROFIT              TOTALPROFIT;
 22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
 23  ;
 24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
 25                             -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
 26  AVAILABLE(RESOURCE).. SUM(PROCESS,RESOURUSE(RESOURCE,PROCESS)
****          $140
**** 140  Unknown symbol
 27                       *PRODUCTION(PROCESS))  =L= RESORAVAIL(RESOURCE);
 28  MODEL RESALLOC /ALL/;
```

6.8.13 Duplicate names - error N

Multiple declarations of items with the same name will cause message $195. The example **shorterr18.gms** illustrates such errors under the use of the errmsg=1 option which repositions the error

© 2022 Prof. Bruce McCarl

message explanatory text.

```
  8  PARAMETER PRICE(PROCESS)      PRODUCT PRICES BY PROCESS
  9               /makechair 6.5 ,maketable 3, makelamp 0.5/
 10           Yield(process)  yields per unit of the process
 11               /Makechair 2   ,maketable 6 ,makelamp 3/
 12         PRODCOST(PROCESS)     COST BY PROCESS
 13               /Makechair 10  ,Maketable 6, Makelamp 1/
 14         RESORAVAIL(RESOURCE)  RESOURCE AVAILABLITY
 15              /plantcap 10 ,salecontrct 3/;
 16  TABLE RESOURUSE(RESOURCE,PROCESS) RESOURCE USAGE
 17                     Makechair   Maketable  Makelamp
 18      plantcap          3           2          1.1
 19      salecontrct       1          -1;
 20  POSITIVE VARIABLES PRODUCTION(PROCESS) ITEMS PRODUCED BY PROCESS;
 21  VARIABLES          PROFIT              TOTALPROFIT;
 22  EQUATIONS          OBJT OBJECTIVE FUNCTION ( PROFIT )
 23                     RESORAVAIL(RESOURCE) RESOURCES AVAILABLE ;
****                             $195
**** 195  Symbol redefined with a different type
```

6.8.14 Referencing item with wrong set - error O

When an item is reference over a different set than it is defined over and that set is not a subset of the original set one gets a domain error **$171**. The example **shorterr19.gms** illustrates such errors under the use of the errmsg=1 option which repositions the error message explanatory text.

```
 23                     AVAILABLE(RESOURCE) RESOURCES AVAILABLE ;
 24  OBJT.. PROFIT=E=   SUM(PROCESS,(PRICE(PROCESS)*yield(process)
 25                             -PRODCOST(PROCESS))*PRODUCTION(PROCESS)) ;
 26  AVAILABLE(process).. SUM(resource,RESOURUSE(RESOURCE,PROCESS)
****                    $171
**** 171  Domain violation for set
 27                        *PRODUCTION(PROCESS))  =L= 10;
 28  MODEL RESALLOC /ALL/;
 29  SOLVE RESALLOC USING LP MAXIMIZING PROFIT;
****                                        $257
**** 257  Solve statement not checked because of previous errors
 30
 31  parameter solprod(process) report of production;
 32  solprod(resource)= RESORAVAIL(RESOURCE);
****                   $171
**** 171  Domain violation for set
```

6.8.15 ORD on an unordered set - error P

When ORD is used on a set that is not ordered one receives error 198. Ordered sets typically are ones with explicit set elements. Unordered sets generally refer to cases where set elements are calculated or elements appear in different order across different sets. The last case is illustrated in

© 2022 Prof. Bruce McCarl

the following two examples errorwithord.gms

In the first example we define a set that uses elements that have already been defined in another set that has some of the elements in the first one but is a more extensive set

```
set a    a couple of the elements /r2,r3/;
set b    more elements /r1*r4/;
scalar count counter /0/;
loop(b$(ord(b) gt 3),count=count+1);
display count;
```

In this case if one printed out b it would be
r2 r3 r1 r4
and GAMS will produce the error. To fix this one could just reorder the two set definitions defining b before a.

In the second example we define a set that uses elements that have already been defined in another set but in a different order.

```
set a1a all the elements /r1*r10/;
set a1b  elements in different order /r3,r1/;
loop(a1b$(ord(a1b) gt 3),count=count+1);
display count;
```

To fix this one must always specify the elements in set 1b in the same order as in 1a. This is required because of the Unique Element List and associated ordering employed within GAMS.

7 More Language Basics

This section covers a slightly advanced set of GAMS language elements. The coverage is organized by chapter with the chapters covering:

Rules for Item Names, Element Names and Explanatory Text
Including Comments
Calculating Items
Improving Output via Report Writing
Rules for Item Capitalization and Ordering
Conditionals
Control Structures

© 2022 Prof. Bruce McCarl

7.1 Rules for Item Names, Element Names and Explanatory Text

There are rules imposed in GAMS relative to the maximum size and composition of item names, set element names and explanatory text. They are discussed below by category.

Item name rules
Element name rules
Explanatory text rules

7.1.1 Item name rules

The statements declaring sets, scalars, parameters, tables, files, acronyms, variables, equations and models all name items. The names for these items must follow a set of rules. The name of any item

- Can be up to 63 characters long, if not using MPSGE.
- Must start with an alphabetic character.
- Can contain both alphabetic and numeric characters.
- Cannot contain spaces.
- Cannot contain special characters excepting an _ (underscore).
- Can be typed and referenced in a case insensitive manner but the case structure used in the first occurrence of the name will be the way it is printed out.
- Should in my opinion for readability and definition be long enough so that the names are explanatory as argued in the Writing Models and Good Modeling Practices chapter.
- MPSGE is limited to 10 character set names. Also if you use a set to index subsets in an CES function, the limit if 4 characters.

Examples:

Below the item names are in **red** (**namerules.gms**)

```
Variable    A6item;
Set        shoes        /tennis,dress/;
Parameter here_is_a_long_name       /1/;
Equation    Whererididiputmy(shoes);
Acronym     Monday, a6data, a7_it;
```

7.1.2 Element name rules

Set or Sets statements declare names for individual set elements. These element names must follow a set of rules. An element name

- Can be up to 63 characters long.
- Should try to attain unique identification in the first 10 characters since longer names will be truncated to 10 characters in certain displays.
- May be quoted or unquoted where:

© 2022 Prof. Bruce McCarl

- In unquoted elements
  - Must start with an alphabetic or numeric character.
  - Spaces are not allowed.
  - The only special characters allowed are +,-,_
  - Cannot be the same as a GAMS reserved words (set, table, variable, parameter, etc.) are not allowed.
- In quoted elements
  - Spaces are allowed.
  - Any special characters are allowed (including international ones if charset=1 option used).
  - Must have quotes that begin and end each element name.
  - Use either the symbol " or ' for the quotes.
  - GAMS reserved words are allowed in quotes.
- Quoted and unquoted elements can be mixed.

- Is case sensitive in subsequent references but case structure used in the first occurrence of the name will be the way it is printed out.
- Will be referred to in quotes in assignments or equations if an individual element is to be addressed.

Examples:

Here the element names are in blue. Note for ease of understanding, each element is on a separate line (**namerules.gms**)

```
Set   mshoes /       tennis
                     has+special-characters
                     an_underscore
                     3
                     3number        /;
Set   shoes2 /       "variable"
                     "*starts_with_*"
                     "has a space"
                     'has "quotes" in it'
                     'contains,a comma'   /;
Set   shoes3 /       tennis
                     "has spaces"         /;
 x("Tennis")=3;
y('an_underscore')=1;
```

7.1.3 Explanatory text rules

The statements declaring sets, scalars, parameters, tables, files, acronyms, variables, equations and models all can contain explanatory text. These explanatory text entries must follow a set of rules. Namely they

© 2022 Prof. Bruce McCarl

- Are optional
- Can be up to 255 characters long
- May be quoted or unquoted
- May contain spaces
- Must all be entered on one line along with the item name
- In unquoted text
  - Can only contain the special character _ (underscore).
  - Cannot contain GAMS reserved words (variable, parameter, etc.).
- In quoted elements
  - Any special characters are allowed (including international ones if charset=1 option used)
  - Must have matching quotes at the beginning and end of each element name.
  - May use either " or ' but the usage of these must match up.
  - May contain GAMS reserved words are allowed in quotes.
  - Can nest quiotes with oly the outer quotes being of concern to GAMS i.e. they could contain "A nested 'quote' like this"
- Should not be treated as optional for named items since will appear in the Symbol List as argued in the Writing Models and Good Modeling Practices chapter.

Note the explanatory text for set elements is only accessible through the .te put file command

Examples:

Below the explanatory text for item names in **green** and the explanatory text for set element names in **blue**. Note for ease of understanding, each example has a separate line for each element (**namerules.gms**)

```
Set             shoess    "my test"
                        /tennis  here i can tell you what this is
                         dress   'special characters if needed /*+-'/
Set             kshoes        with underscore_;
Set             kshoes3       "with special characters * / ? , ";
Set             kshoes4       "with 'quotes' like in can't and won't "
Parameter       data3           THIS IS explanatory text       /1/;
Variable        A6item        "***** look at this one";
Equation        Wh(shoes)     Model equation;
Model           notexthere /all/;
```

7.2 Including Comments

GAMS code can have comments inserted within it in several ways. A description of each follows.

© 2022 Prof. Bruce McCarl

Blank lines
Single line comments
Multiple line comments
End of line comments
In line comments
Outside margin comments
Hidden comments

7.2.1 Blank lines

One can freely enter blank lines to set off certain sections and enhance readability. For example in **commentdol.gms** I have 2 blank lines.

```
set a /a1,a2/;
set b /a2,c2/;
set c /a3,d3/;

parameter d(a,b,c);
d(a,b,c)=1.234567;

option d:1;display d;
option eject;
```

7.2.2 Single line comments

Users may insert a single line comment (or a group thereof) on any line by placing an asterisk in column 1 followed by any text whatsoever. The commented text is completely ignored by the GAMS compiler. It can contain GAMS reserved words, messages or any other content.

Uses can cause GAMS to recognize a character other than an asterisk as the column 1 delimiter. This is done by using the $ command $Comment c where the character c replaces * as the comment delimiter. For example

```
$comment !
```

changes the character to a !. An example follows and is in **commentdol.gms**

```
*normal comment
*next line is deactivated GAMS statement
*  x=sum(I,z(i));
$comment !
!comment with new character
```

Notes:

- Comments may be intermixed with other GAMS instructions and may be used anywhere in the GAMS code.

© 2022 Prof. Bruce McCarl

- The * or other character must appear in column 1 or it (the *) will be treated as a multiplication symbol.
- The character used in redefinition of the * comment identifier must be chosen carefully as that character is thereafter prohibited from ever appearing in column 1.
- A multiplication symbol can never appear in column 1 while * is the comment beginning character.
- One line comments appear in the echo print as numbered lines.

7.2.3 Multiple line comments

While one can enter a series of comments beginning with an * in column 1, there are cases where it is most convenient to incorporate a multi line text comment without the need for an * in each line. This is done by using the paired $ commands $Ontext and $Offtext. $Ontext causes all lines after it to be treated as comments regardless of content. $Offtext terminates the comment begun by a $Ontext making subsequent lines regular GAMS statements. The lines in between constitute the comment.

Example:

(commentdol.gms)

```
$ontext
Here is a file showing how to use
various types of comments

valid GAMS statements can also be in the
comment but will not be compiled

set a /1*11/;
$offtext
set a /a1,a2/;
set b /a2,c2/
set c /a3,d3/;
```

Notes:

- Lines in the $ontext $offtext sequence are copied through to the echo print listing but without line numbers.
- These items can be used to rapidly deactivate large sections of code that one does not wish to execute or compile as in speed or memory use searches.

7.2.4 End of line comments

Comments may be included on the end of lines containing GAMS code. This involves both activation and specification.

- Activation - while GAMS accommodates such statements it must be told to do so. This is done by using the command $Oneolcom.
- Specification -- end of line comments are set off through the entry of the delimiting character string !! that renders the remaining part of the statement as a comment.

© 2022 Prof. Bruce McCarl

- One can change the activating character string using the command $eolcom.

Example:

(commentdol.gms)

```
$oneolcom
x=x+1;       !! eol comment
x = x    !! eol comment in line that continues on
   +1;
$eolcom &&
x=x+1;       && eol comment with new character
```

Notes:

- Before end of line comments can be used a $oneolcom activating statement must appear.
- End of line comments are copied through to the echo print listing on the appropriate lines.
- The lines on which end of line comments appear must be fully valid GAMS statements but the comment can appear on intermediate lines of a statement that terminates later.
- The ability to add an end of line comment can be terminated by entering the command $Offeolcom.
- One can redefine the activating characters from !! to some other choice using the command $Eolcom cc where cc is the new character string delimiting beginning of a comment as illustrated above.

7.2.5 In line comments

Comments may be included intermixed with instructions in a line of GAMS code. This involves both activation and specification.

- Activation -- while GAMS accommodates such statements it must be told to do so. This is done by using the command $Oninline.
- Specification -- in line comments are set off through the entry of the character string /* before the comment and */ after the string which renders all the characters within these delimiters as a comment.
- Activating characters can be changed with the command $inlinecom.

Example:

(commentdol.gms)

```
$oninline
x=x      /* in line comment*/ +1;
x = x  /* in line comment in line
          that continues on */
   +1;
$inlinecom /&  &/
x=x      /& eol comment with new character &/ +1;
```

© 2022 Prof. Bruce McCarl

Notes:

- Before in line comments can be used a $Oninline activating statement must appear.
- In line comments must be set off before and after with the initiating (/*) and terminating delimiters (*/).
- In line comments are copied through to the echo print listing on the appropriate lines.
- The lines in which in line comments appear must be fully valid GAMS statements, but the comment can appear on intermediate lines of a statement that terminates later.
- In line comments once begun may span multiple lines before the terminating character appears.
- The ability to add an in line comment can be terminated by entering the command $Offinline.
- One can redefine the initiating and terminating characters from /* and */ to some other choice using the command $Inlinecom cc dd where cc and dd are the new delimiting character strings designating beginning and end of a comment as illustrated above.Inline comments may be allowed to be nested using $Onnestcom.

7.2.6 Outside margin comments

Comments may also be included on the front and back of lines with GAMS code. This is done by first defining the columns where the active GAMS instructions appear. Subsequently one enters comments immediately preceding or succeeding the active code columns. In particular, one can specify the first and last columns on a page of GAMS code that are active GAMS code and thereby both

- A leading area of each line can be set off so that the GAMS compiler will ignore it. This is specified by setting the left most column where valid GAMS instructions can occur using the command $Mincol cc. There cc is the first active column where GAMS code appears. In turn, anything to the left of that column is treated as a comment.
- A trailing area of each line can be set off so that the GAMS compiler will ignore it. This is specified by setting the right most column where valid GAMS instructions can occur using the command $Maxcol cc. There cc is the last active column where GAMS code appears. In turn anything to the right of that column is treated as a comment.

Example:

(margin.gms)

```
$ontext
         1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901234567890
$offtext
$mincol 20 maxcol 45
Now I 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
```

Only the black section of the statements are active since they appear between columns 19 and 45, and

© 2022 Prof. Bruce McCarl

anything before 19 column or after column 45 is treated as a comment.

Notes:

- Before comments set off by margins can be used either a $mincol or $maxcol statement must be used.
- Mincol defaults to 1 and is not specified through a $mincol command means there will be no beginning inactive content on GAMS lines.
- Maxcol defaults to the maximum line length currently 32767and if not specified through a $maxcol command means there will not be trailing inactive content on GAMS lines.
- Setting Maxcol to 0 causes GAMS to set it to the maximum allowable number of columns.
- The full content of the lines are copied through to the echo print listing including the area below the mincol and above the maxcol.
- The GAMS statement components falling within the specified columns mincol through maxcol must be fully valid GAMS statements.
- Delineators setting off the margins are entered through the use of $Onmargin and can be removed through $Offmargin. An example appears there.

7.2.7 Hidden comments

Users may insert a hidden single line comment (or a group thereof) at any point in the code by placing a $Hidden command followed by comment text on the same line. The commented text is completely ignored by the GAMS compiler, but is not copied to the echo print component of the LST file. It can contain GAMS reserved words, messages or whatever. An example follows and is in **commentdol.gms**

```
$hidden a comment I do not want in LST file
set a /a1,a2/;
set b /a2,c2/
set c /a3,d3/;
```

7.3 Calculating Items

Calculations are inherent in GAMS replacement statements and in equation specification (..) commands.

Basic components of calculations
Operations over set dependent items
Items that can be calculated
Cautions about calculations
Potential other components in calculations
Including conditionals

7.3.1 Basic components of calculations

Calculations contain basic components in terms of algebraic specification and in terms of operators that

© 2022 Prof. Bruce McCarl

act over sets.

Operators
=
.. statements
Basic arithmetic + - * / **
Arithmetic hierarchy
Changing hierarchy using parentheses

7.3.1.1 Operators

The basic operators in equations are the

- = equal sign,
- .. equation definition
- +, -, *, / and ** operators that allow addition, subtraction, multiplication, division and exponentiation and
- (, { and [which are code grouping parentheses.

These are discussed below plus their hierarchy in calculation.

=
.. statements
Basic arithmetic + - * / **
Arithmetic hierarchy
Changing hierarchy using parentheses

7.3.1.1.1 =

All replacement (also called assignment) statements in GAMS contain an = sign. The basic format is (**calculate.gms**)

```
x=4;
Z(i)=12;
Y(i)$z(i)=z(i);
Q(i)=yes;
```

where the scalar or set dependent item on the left hand side is set equal to the scalar or expression on the right hand side. The term on the left hand side must always be a single scalar, parameter or set item potentially indexed over sets. The term can also contain a conditional. The term on the right hand side is a constant, expression, acronym or special value.

7.3.1.1.2 .. statements

Calculations can appear in .. statements as discussed in the Variables, Equations, Models and Solves chapter.

© 2022 Prof. Bruce McCarl

```
Eq1[I].. zZ(i)=e=12+x;
Eq2[I].. zZ(i)=e=12-x;
```

7.3.1.1.3 Basic arithmetic + - * / **

Commonly it is desirable to add, subtract, multiply, divide or exponentiate items in replacement statements. This is done employing the symbols + - * / **. Examples follow (**calculate.gms**)

```
x=4+6+sqrt(7);
Z(i)=12+x;
Eq1[I].. zZ(i)=e=12+x;
Y(i)=+z(i);
x=x-3-sqrt(7);
Z(i)=12-x;
Eq2[I].. zZ(i)=e=12-x;
Y(i)=-z(i);
x=x-3-sqrt(7);
Z(i)=12*x;
Eq3[I].. zZ(i)=e=12*x;
Y(i)=44*z(i);
x=x-3/sqrt(7);
Z(i)=12/x;
Eq4[I].. zZ(i)=e=12/x;
Y(i)=1/z(i);
x=3**sqrt(7);
Z(i)=12.0**x;
Y(i)$(z(i)-1 gt 0)=z(i)**0.5;
Y(i)=14*x**2/4-3+2;
```

A + - * / or ** cannot go on the left hand side unless it is in the numerical evaluation of a Conditional.

These operators may be used with sets as described below or in the set arithmetic part of the Sets chapter.

7.3.1.1.4 Arithmetic hierarchy

Within GAMS replacement statements are evaluated in a three-step hierarchy (unless they are enclosed in parentheses). That hierarchy is

Operator	Description	Priority
**	exponentiation	1
* /	multiplication and division	2
+ -	addition and subtraction	3

The priority 1 items are done before the priority 2 and then the 3. However, items in parentheses or brackets are always resolved first. But within the parentheses the above hierarchy is followed. When

© 2022 Prof. Bruce McCarl

multiple items appear of the same priority resolution is from left to right. Calculation hierarchy also involves logical conditions as fully specified in the Conditional notes.

7.3.1.1.5 Changing hierarchy using parentheses

Items within parentheses or brackets are resolved before doing numerical calculations and the innermost parentheses are resolved first. Furthermore, any of the pairs () {} or [] can be used. For example, in the following

```
Z{I}=+{34/(11+12)}+11;
eq[i].. zZ{I}=l=10*[3+2]**{34/(11+12)}+11-1;
```

the 11+12 is computed first then the other parentheses or brackets surrounded terms.

7.3.2 Operations over set dependent items

Numerical operations can involve sets either operating over them or doing operations with them. Fundamentally these involve sums, finding the biggest or smallest number in a set, multiplying elements and doing calculations involving set elements. It is also worthwhile discussing forms of set referencing in these items.

= replacement or .. statements
Sum , Smax, Smin, Prod
Alternative set addressing schemes

7.3.2.1 = replacement or .. statements

When statements like

```
x(i)=4;
Eq4[I].. zZ(i)=e=12/x(i);
```

are included in a GAMS code all of the cases associated with each and every element in I are computed. The order that the calculations or done in terms of the elements of the sets depends on the order of the set elements as discussed in the Rules for Item Capitalization and Ordering chapter.

7.3.2.2 Sum , Smax, Smin, Prod

Frequently in a program one wishes to compute items over the set elements associated with parameters.

7.3.2.2.1 Sum

One might wish to sum items as follows

```
r=sum(I,y[I]);
r=sum((I,j),y(i)+w(j));
eq5[I]..  zz[I]=e=sum[j,zz[i-1]+zq[j]];
```

© 2022 Prof. Bruce McCarl

Here the sum is computed over all of

the elements in I for the first equation
the elements in I and j for the second equation
the elements in j for the third equation

Notes:

- The general syntax is sum(settovary,expression) where
  - The settovary is the name of the sets or sets that will be varied
  - When more than one set is to be varied they are enclosed in parentheses – sum((i,j),x(i,j)).
  - Expression is a generally a function of the set in the sum
  - A (occurs just after the word sum and matches with a) at the end of the sum.
- In replacement statements parameters, and scalars can freely appear. Variables and equations can only be present if attributes of them are being addressed.
- In model equation specification statements parameters, scalars and variables can freely appear.
- The expression after the comma is fully computed and then applied to the sum as if the term were in parentheses.

7.3.2.2.2 Smin Smax

One might wish to find the largest or smallest values over a set. This is done using

```
r=smax(I,y[I]);
r=smax((I,j),y(i)+w(j));
```

Here the maximum element from the set is returned back over all of

the elements in I for the first equation
the elements in I and j for the second equation

Use of the syntax smin will find the smallest element across the set.

```
r=smin(I,y[I]);
r=smin((I,j),y(i)+w(j));
```

Notes:

- The general syntax is smax(settovary,expression) where
  - the settovary is the name of the sets or sets that will be varied
  - When more than one set is to be varied they are enclosed in parentheses – smin((i,j),x(i,j)).

© 2022 Prof. Bruce McCarl

- expression is a generally a function of the set in the smax
- a (occurs just after the word smax/smin and matches with a) at the end of the smax/smin.

- In replacement statements only parameters, scalars, and table data may freely appear. Variables and equations can only be present if attributes are being addressed.
- You cannot generally use smin and smax in equations for models unless you are dealing with a DNLP type of model.
- The expression after the comma is fully computed and then applied to the calculation as if the term were in parentheses.

7.3.2.2.3 Prod

One might also wish to compute the product of items across a set as follows

```
r=prod(I,y[I]);
r=prod((I,j),y(i)+w(j));
eq5[I]..  zz[I]=e=prod[j,zz[i-1]+zq[j]];
```

Here the product is computed over all of

the elements in I for the first equation
the elements in I and j for the second equation
the elements in j for the third equation

Notes:

- The general syntax is prod(settovary,expression) where
  - the settovary is the name of the sets or sets that will be varied
  - When more than one set is to be varied they are enclosed in parentheses – prod((i,j),x(i,j)).
  - an expression is a generally a function of the set in the prod
  - a (occurs just after the word prod and matches with a) at the end of the prod.
- In replacement statements only parameters, scalars, and table data may appear. Variables can only be present if attributes of them are being addressed.
- In model equation specification statements all the data items plus variables can appear.
- The expression after the comma is fully computed and then applied to the product as if the term were in parentheses.

7.3.2.3 Alternative set addressing schemes

Sometimes it is not desirable to sum etc. over all the members of a set but only selected items.

7.3.2.3.1 Avoiding set domain errors

When summing etc over sets one must control all the sets in the expression or a domain error arises.

© 2022 Prof. Bruce McCarl

Sets are controlled in some mixture of the following four ways (**calculate.gms**).

- Sets may be completely operated over by the sum, prod etc.

```
mZ=sum(I,mr(i)*xvar.l(i));
Obj.. z22=e= sum(I,mr(i)*xvar(i));
```

- Sets may be operated over by the equation definition

```
mX(Zzz)=sum(I,ar(zzz,i)*ma(i));
Resource(j).. z2(j)=e= sum(I,jr(I,j)*xvar(i));
```

- Sets may be operated over by Control Structures (loop, for, etc.)

```
Loop(zzz, z22=z22+sum(I,ar(zzz,i)*mr(i)));
```

- Specific set elements may be specified.

```
Z22=sum(I,ar('z1',i)*mr(i));
```

7.3.2.3.2 Multiple sets

Operations indexing sets may involve multiple sets. In such a case one can write the function in multiple ways. For example the sum over the sets I and J of c(I,J)*v(I,J)) can be written as either (**calculate.gms**)

```
R=sum((I,J),c(I,J)*v(I,J));
```

or

```
R=sum(I,sum(J,c(I,J)*v(I,J)));
```

Notes:

- Note how the parentheses match up.
- When multiple sets are addressed they need to be enclosed in parentheses.
- PROD, SMIN and SMAX indices are handled just as the SUM is above.

7.3.2.3.3 Conditionals to restrict set coverage

In sums, minimums, maximums and products it is not always desirable to operate over all the members of a set. One accomplishes this by adding a conditional or addressing over a tuple or subset as discussed below. Such a conditional can take on a number of forms.

Examples:

(calculate.gms)

Examples using a relatively simple conditional on the numerical value of a parameter are

```
mA(i)$xvar.l(i)=mr(i);
mA(i) =mr(i) $xvar.l(i);
r=sum(i$xvar.l(i),mr(i));
```

© 2022 Prof. Bruce McCarl

```
r=smin(i$xvar.l(i),mr(i));
r=prod(i$xvar.l(i),mr(i));
Eq7(i)$ma(i).. zZ{I}=l=10*[3+2]**{34/(11+12)}+{11-1};
Eq8(j)$w(j)..  z2(j)=e= sum(I$ma(i),jr(I,j)*xvar(i)) ;
```

Notes:

- A fuller discussion of these topics appears in the conditional and sets chapters.
- The equations with the conditionals on the left hand side operate significantly different from those with the conditional on the right hand side as discussed below.

7.3.2.3.4 Tuples and subsets to restrict set coverage

In sums, minimums, maximums and products it is not always desirable to sum etc. over the all the members of a set. One accomplishes this by adding a conditional as discussed immediately above or addressing over a multiple dimensional set (a tuple) or a subset. Such addressing can take on a number of forms.

Examples:

(calculate.gms)

```
mA(i)$mysubset(i)=mr(i);
ma(mysubset(i))=mr(i);
mA(i) =mr(i) $mysubset(i);
mA(i) =sum(atuple(I,j),v(I,j));
v(i,j)$atuple(i,j)=3;
v(atuple(i,j))=3;
v(atuple)=3;
r=sum(mysubset,mr(mysubset));
r=sum(mysubset(i),mr(i));
r=sum(mysubset(i),mr(mysubset)+mr(i));
r=sum(atuple(I,j),v(I,j));
r=smin(i$mysubset(i),mr(i));
r=smax((I,j)$atuple(I,j),v(I,j));
r=smax(atuple(I,j),v(I,j));
Eq11(i)$mysubset(i).. zZ{I}=l=10*[3+2]**{34/(11+12)}+{11-1};
Eq12(mysubset).. zZ{mysubset}=l=10*[3+2]**{34/(11+12)}+{11-1};
Eq13(mysubset(i)).. zZ{I}=l=sum(atuple(I,j),ord(i)+ord(j));
Eq14(atuple(i,j)).. 1=e= twovar(i,j) ;
Eq15(I,j)$atuple(i,j).. 1=e= twovar(i,j) ;
eq16(j).. 1=e=sum(atuple(I,j),v(I,j)*twovar(i,j));
```

Notes:

- Additional discussion of these topics appears in the conditional and sets notes.
- Equations with the conditionals on the left hand side operate significantly different from those with the conditional on the right hand side as discussed below.

© 2022 Prof. Bruce McCarl

- Use of the tuple as the set summed over results in the component sets being varied as in sum (atuple(I,j),v(I,j)) where both I and j are varied. But the only cases of I and j that are considered are those that are jointly in the tuple.
- Use of the tuple in an equation defined over some of the sets in the tuple results in the component sets not in the equation definition being varied as in mA(i) =sum(atuple(I,j),v(I,j)) where the j case that are in the tuple associated with the particular I case that is being computed are varied.
- Use of the subset and the superset name in the sum gives access to both in an equation as in r=sum(mysubset(i),mr(mysubset)+mr(i)); where both the sets I and mysubset are accessible.
- When all items in a tuple are to be acted over one can use syntax like v(atuple)=3; which will work even if v is defined over multiple sets as long as the tuple is defined over those same sets.

7.3.3 Items that can be calculated

When calculating one can compute sets, data, items in equations or acronyms. I discuss each below.

Sets
Data
Equation calculations
Acronyms

7.3.3.1 Sets

Set elements may be calculated as discussed in the Sets chapter. This involves setting elements of subsets to

- Yes to activate them
- No to remove them
- Using set arithmetic to form unions, intersections, complements etc.

The most common calculation involves definition of subset elements. For example to make every element of a superset appear in a subset one would use the command (**calculate.gms**)

```
Subset(superset)=yes;
```

while removing all is done using

```
Subset(superset)=no;
```

Tuples are also commonly computed based on data as illustrated below:

```
Cantravel(origin,destination)
      $(distance(origin,destination) gt 0)=yes;
```

7.3.3.2 Data

Data calculations refer to the case where elements of a parameter or scalar are computed in a

© 2022 Prof. Bruce McCarl

statement involving an = as illustrated above. Note these calculations are static in that the calculation is only performed at the point it appears in the code and is not updated when any input data are changed. For example consider the case below (**calculate.gms**)

```
r=1;
s=2*r;
Display 'one',s;
r=2;
display 'two',s;
```

Note in this case the value of s is unchanged between display one and display two even though the r parameter is changed. This means one needs to reissue the calculations if one is to update calculations when the input data to that calculation is revised. More will be said about this in the section on static/dynamic calculations below.

7.3.3.3 Equation calculations

Equation calculations refer to the case where terms in a model equation are computed in a statement involving a .. command for a named equation. Note these calculations are "dynamic" being updated every time a Solve statement is executed. More will be said about this in the section on static/dynamic calculations below.

7.3.3.4 Acronyms

Assignment statements can involve acronyms.

Examples:

(acronym.gms)

```
acronyms nameforit,nextone;
acronym  acronym3
acronym  doit
parameter textstrings(i)
     /i1 nameforit,       i2 nextone,      i3 acronym3/ ;
parameter textstring(i)
     /i1 doit,      i2 1,      i3 doit/ ;
display textstrings ;
parameter zz(i);
zz(i)= textstring(i)  ;
display zz;
```

Notes:

- Assignment statements involving acronyms can only set other items equal to acronyms or other parameters containing acronyms.
- Numerical operations (+ - * / **) cannot be done.
- Numbers can be mixed in with acronyms in a data item but cannot be subsequently manipulated

© 2022 Prof. Bruce McCarl

numerically.

- Acronyms cannot be used in numerical terms in equation specification (..) commands excepting where they appear in conditionals.

7.3.4 Cautions about calculations

GAMS treats calculations in three distinctly different ways.

Dynamic
Static
Repeated static
Cautions about dynamic /static calculations

7.3.4.1 Dynamic

Dynamic calculations repeated every time the model is generated. Only calculations in the model .. statements are dynamic

```
Obj..       ObjF=E=SUM(Crop,Acres(CROP)*
               (Price(CROP)*Yield(CROP)-Cost (CROP)));
```

7.3.4.2 Static

Calculations executed once only at the place the GAMS instruction appears in the code.

```
Revenue (Crop)= Price(Crop)*Yield(Crop)-Cost(Crop);
x.up(crop,region)=(1+flex)*baseacre(crop,region);
x.scale(crop,region)=scalefac(crop);
```

7.3.4.3 Repeated static

These are calculations within a repeatedly executed GAMS control structures (loop, for, while) but are static within that control structure such as

```
LOOP ( LANDCHANGE,
   LAND=LAND*
      (1+VALUE(LANDCHANGE)/100.);
       solve farm using lp maximizing income)
```

7.3.4.4 Cautions about dynamic /static calculations

The types of calculations are very important for 2 reasons

- When static or repeated static calculations are present and some of the calculation input data are revised between the point where the calculations executed in the model solve statement appears then you must repeat the static calculations if you want the results to be current
- When you have repeated static calculations then the data changes may buildup and you may need to reset the data to base levels if you want your data always to revert back to the base

© 2022 Prof. Bruce McCarl

case

Examples:

(nondyn.gms)

Below the first two solves are identical even though the Price parameter is altered. Why? The answer is because the calculation for Revenue is not updated after the Price change. The repeated Revenue calculation and second solve corrects this.

```
set crop /corn/
parameter price(crop),yield(crop),cost(crop),revenue(crop);
Price(Crop) = 2.00;
Yield(Crop) = 100;
Cost(Crop) = 50;
Revenue (Crop) = Price(Crop)*Yield(Crop)-Cost(Crop);
Equations
obj         objective function
Land              Land available;
Positive Variables      Acres(Crop) Cropped Acres
Variables         Objf  Objective function;
obj..       objf=E=Sum(Crop, Revenue(Crop)*Acres(Crop));
Land..            Sum(Crop, Acres(Crop))=L=100;
Model       FARM/ALL/
SOLVE FARM USING LP Maximizing objf;
Price ("corn")=2.50;
SOLVE FARM USING LP Maximizing objf;
Revenue (Crop)= Price (CROP)*Yield(Crop)-Cost(Crop);
Solve FARM Using LP Maximizing objf;
```

One can revise the model so it contains a dynamic calculation that eliminates the problem. Namely, computing revenue in the objective function corrects this. (see the bottom of **nondyn.gms**)

```
Obj..       ObjF=E=SUM(Crop,(Price(CROP)*Yield(CROP)
                        -Cost (CROP))Acres(CROP);
```

Note one can go too far with this making a model too complex.

Another example is also in order on data preservation/cumulative data changes in the case of repeated static calculations. (**repstatic.gms**)

GAMS lives for the moment, when a calculation is issued then all prior values are overwritten for calculated items.

```
SCALAR LAND /100/;
PARAMETER SAVELAND;
SAVELAND = LAND;
SET LANDCHANGE  SCENARIOS FOR CHANGES IN LAND
/R1,R2,R3/
PARAMETER
```

© 2022 Prof. Bruce McCarl

```
VALUE(LANDCHANGE) PERCENT CHG IN LAND
                        /R1 +10 , R2 + 20 , R3 +30/
LOOP ( LANDCHANGE,
       LAND = LAND * (1 + VALUE ( LANDCHANGE ) / 100. )
       display "without reset" ,land);
```

Running this model yields

```
----      9 without reset
----      9 PARAMETER LAND      =      110.000 = 100*1.1
----      9 PARAMETER LAND      =      132.000 = 100*1.2*1.1
----      9 PARAMETER LAND      =      171.600 = 100*1.3*1.2*1.1
```

Here I see cumulative changes and need to ask if they are intended.

If you want to revert to the original values then you have to instruct GAMS to do that. Replace the loop with (**repstatic.gms**)

```
LOOP ( LANDCHANGE,
      LAND=saveLAND*(1+VALUE ( LANDCHANGE ) / 100. )
      display "based on saveland" ,land);
```

or

```
LOOP ( LANDCHANGE,
       land=saveland;
       LAND = LAND * (1 + VALUE ( LANDCHANGE ) / 100. )
       display "with saveland reset", land);
```

Both yield

```
----   12 PARAMETER LAND     =      110.000 = 100*1.1
----   12 PARAMETER LAND     =      120.000 = 100*1.2
----   12 PARAMETER LAND     =      130.000 = 100*1.3
```

Generally if you are changing items between solves, then reset them to base values to avoid accumulating changes.

7.3.5 Potential other components in calculations

Calculations can contain a variety of GAMS items that are not parameters or numbers. Here I review the possibilities for inclusion of logical expressions, functions, special values, variable attributes, model attributes and sets intermixed with numbers.

Mixing logical expressions, sets and numbers
Functions
Special values
Model and optimal solution items

7.3.5.1 Mixing logical expressions, sets and numbers

One can mix logical expressions and sets with numerical calculations. Namely, if a logical item is

© 2022 Prof. Bruce McCarl

included in a numerical calculation, it takes on a value of one if the result is true and zero if it is false. Similarly a set is treated as a logical item and returns a 1 if an element is defined and zero otherwise. Items that are Sets can also be equated to numbers and get a yes value if the item is nonzero and no otherwise. The following example illustrates (**mixlogical.gms**)

```
X=1;
z = 100*(x < 4) + ( 3 < 3);
set j(i);
j(i)=x+1;
parameter zz(i);
zz(i)=j(i)*100;
```

Note z evaluates to 100 since only the logical condition on the left is true. Note that this is different from the assignment below,

```
z = (x < 4) or ( 3 < 3);
```

where z evaluates to 1 due to the or operator behaving as explained in the conditional notes.

7.3.5.2 Functions

There are a number of functions available in GAMS. The built-in (intrinsic) functions and the extrinsic functions which can be imported from an external library into a GAMS model.

Intrinsic functions
Extrinsic functions

7.3.5.2.1 Intrinsic Functions

Intrinsic Functions are functions provided by GAMS.
There are commonly used standard functions, mathematical functions, logical functions, time and calendar functions, GAMS utility and performance functions and string manipulation functions.

Common mathematical functions
Other mathematical functions
Logical functions
Time and calendar functions
GAMS utility and performance functions
String manipulation functions: Ord, Ordascii, Ordebcdic
String manipulation functions: Card
Computer Characteristic functions

7.3.5.2.1.1 Common mathematical functions

A number of mathematical functions can be used in numerical expressions.

Abs

© 2022 Prof. Bruce McCarl

Execseed
Exp
Ifthen
Log, Log10, Log2
Max , Min
Prod
Round
Smin , Smax
Sqr
Sqrt
Sum

Expressions can contain a function that calculates the absolute value of an expression or term (**function.gms**).

```
X=abs(tt);
X=abs(y+2);
Eq1..   z=e=abs(yy);
```

This function is not continuous and not smooth. It can be used on data during GAMS calculation or in models on variables or parameters. It's use in .. equation specifications on terms involving variables requires the model type be DNLP, CNS, MCP or some other form that accepts nonsmooth functions.

Expressions can be used to reset and save the seed for the random number generator. Setting a scalar quantity to the function

```
cc=execseed;
```

saves the random number seed that will be used to generate the next random number. In the process, it also re-initializes the random number generator using this seed value.

Setting the function to a scalar quantity (that must be integer)

```
execseed=round(cc,0);
```

resets the random number seed.

This item should be used very infrequently on the right hand side of a replacement statement as this forces a re-initialization of the seed for the random number generator.

Expressions can contain a function that calculates the exponentiation e^q of an expression or term (**function.gms**).

```
X=exp(t);
X=exp(y+2);
Eq2..   z=e=exp(yy);
```

© 2022 Prof. Bruce McCarl

This function is continuous and smooth. It can be used on data during GAMS calculation or in models on variables or parameters. It's use in .. equations in terms that involve variables requires the model type be NLP or one of the other types that handles nonlinear functions.

Expressions can contain a function that take on different values depending on a condition.

```
X=ifthen(condition,expressioniftrue,expressioniffalse);
```

where if the condition is true then the function equals expressioniftrue and otherwise equals expressioniffalse. For example (**function.gms**) below if tt=2 then x will be set to 3 otherwise x=4+y.

```
X=ifthen(tt=2,3,4+y);
```

This function can be used on data during GAMS calculation and also in .. equations. When used in .. equations it requires the model type to be DNLP and it is not supported by all available solvers.

Expressions can contain a function that calculates the natural logarithm or logarithm base 10 of an expression or term (**function.gms**).

```
X=log(tt);
X=log(y+2);
Eq3..   z=e=log(yy);
X=log10(tt);
X=log10(y+2);
Eq4..   z=e=log10(yy);
```

These functions are continuous and smooth. They can be used on data during GAMS calculation or in models. It's use in .. equations in terms that involve variables requires the model type be NLP or one of the other types that handles nonlinear functions.

Expressions can contain a function that calculates the maximum or minimum of a set of expressions or terms. (**function.gms**)

```
X=max(y+2,t,tt);
Eq3..   z=e=max(yy,t);
X=min(y+2,t,tt);
Eq4..   z=e= min (yy,t);
```

These functions are not continuous and are not smooth. They can be used on data during GAMS calculation or in models. It's use in .. equations in terms that involve variables requires the model type be DNLP, CNS, MCP or some other form that accepts nonsmooth functions.

Expressions can contain a function that calculates the product of set indexed expressions or terms (**function.gms**).

```
X=prod(I, a(i)*0.2-0.5);
```

© 2022 Prof. Bruce McCarl

```
Eq7..   z=e= prod(I, a(i)*0.2-0.5);
```

This function is continuous and smooth. It can be used on data during GAMS calculation or in models. It's use in .. equations in terms that involve variables requires the model type be NLP or one of the other types that handles nonlinear functions More on prod can be found here.

Data calculation expressions can contain a function that rounds the numerical result of an expression or term. There are 2 variants of the rounding function. The first (**function.gms**)

```
X=round(q);
X=round(12.432);
```

Rounds the result to the nearest integer value.

The second (**function.gms**)

```
X=round(q,z);
X=round(12.432,2);
```

rounds the result to the number of decimal points specified by the second argument.

This function may be used on data during GAMS calculations. It cannot be used in models.

Expressions can contain a function that calculates the minimum or maximum of set indexed expressions or terms (**function.gms**)

```
X=smin(I, a(i)*0.2-0.5);
Eq8..   z=e= smin(I, va(i)*0.2-0.5);
X=smax(I, a(i)*0.2-0.5);
Eq9..   z=e= smax(I, va(i)*0.2-0.5);
```

This function is not continuous and not smooth. It can be used on data during GAMS calculation or in models on variables or parameters. It's use in .. equations in terms that involve variables requires the model type be DNLP or one of the other types that handles nonsmooth functions. More on Smin and Smax can be found here.

Expressions can contain a function that calculates the square of an expression or term. (**function.gms**)

```
X=sqr(t);
X= sqr (y+2);
Eq10..   z=e= sqr (yy);
```

This function is continuous and smooth. It can be used on data during GAMS calculation or in models on variables or parameters. It's use in .. equations in terms that involve variables requires the model type be NLP or one of the other types that handles nonlinear functions.

Expressions can contain a function that calculates the square root of an expression or term.

© 2022 Prof. Bruce McCarl

(**function.gms**)

```
X=sqrt(a('i1'));
X=sqrt(y+2);
Eq11..   z=e= sqrt(yy);
```

This function is continuous and smooth. It can be used on data during GAMS calculation or in models. It's use in .. equations in terms that involve variables requires the model type be NLP or one of the other types that handles nonlinear functions.

Expressions can contain a function that calculates the sum of set indexed expressions or terms (**function.gms**)

```
X=sum(I,a(i));
Eq7..   z=e= sum(I,va(i));
```

This function is linear. It can be used on data during GAMS calculation or in models on variables or parameters. More on Sum can be found here.

7.3.5.2.1.2 Other Mathematical functions

In the following table 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 optional and otherwise default values will be used. Those default values are specified in the function description.

Function	**Endogenous Classification**	**Description**
arccos(x)	NLP	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 MathWorld
arcsin(x)	NLP	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 MathWorld
arctan(x)	NLP	returns the inverse tangent of the argument x where x is a real number and the output is in radians, see MathWorld
arctan2(y,x)	NLP	four-quadrant arctan function yielding arctangent(y/x) which is the

© 2022 Prof. Bruce McCarl

		angle the vector (x,y) makes with (1,0) in radians
Beta(x,y)	DNLP	beta function as discussed in MathWorld
betaReg(x,y,z)	NLP	regularized beta function, see MathWorld
binomial(n,k)	NLP	returns the (generalized) binomial coefficient for n,k ≥ 0
ceil(x)	DNLP	returns the smallest integer number greater than or equal to x
centropy(x,y[,Z])	NLP	cross entropy: x*ln((x+Z) / (y+Z)), default setting: Z = 1e-20
cos(x)	NLP	returns the cosine of the argument x where x must be in radians, see MathWorld
cosh(x)	NLP	returns the hyperbolic cosine of x where x must be in radians, see MathWorld
cvPower(X,y)	NLP	returns X^y for X ≥ 0
div(dividend,divisor)	NLP	returns *dividend/divisor,* undefined for *divisor = 0*
div0(dividend,divisor)	NLP	returns *dividend/divisor,* returns *1e299* for *divisor = 0*
eDist(x1[,x2,x3,…])	NLP	Euclidean or L-2 Norm, see MathWorld, default setting: x2,x3,… = 0
entropy(x)	NLP	entropy: -x*ln(x)
errorf(x)	NLP	calculates the integral of the standard normal distribution from negative infinity to x, see MathWorld
fact(X)	any	returns the factorial of X where X is an integer
floor(x)	DNLP	returns the greatest integer number less than or equal to x
frac(x)	DNLP	returns the fractional part of x
gamma(x)	DNLP	gamma function as discussed in MathWorld
gammaReg(x,a)	NLP	regularized gamma function, see MathWorld
logBeta(x,y)	NLP	log beta function: log(B(x, y))

© 2022 Prof. Bruce McCarl

logGamma(x)	NLP	log gamma function as discussed in Mathworld
mapVal(x)	none	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
mod(x,y)	DNLP	returns the remainder of x divided by y
ncpCM(x,y,Z)	NLP	function that computes a Chen-Mangasarian smoothing equaling: x - Z*ln(1+exp((x-y)/Z)
ncpF(x,y[,Z])	NLP	function that computes a Fisher smoothing equaling: sqrt(x^2 + y^2 + 2*Z) - x - y
ncpVUpow(r,s[,MU])	NLP	NCP Veelken-Ulbrich smoothed min: $ncpVUpow = \begin{cases} \frac{r+s-\lvert t\rvert}{2}, if\ \lvert t\rvert \geq \mu \\ \frac{r+s-\frac{\mu}{8}\cdot(-(\frac{t}{\mu})^4+6(\frac{t}{\mu})^2+3)}{2}, otherwise \end{cases}$ where t=r-s, default setting: MU = 0
ncpVUsin(r,s[,MU])	NLP	NCP Veelken-Ulbrich smoothed min: $ncpVUSin = \begin{cases} \frac{r+s-\lvert t\rvert}{2}, if\ \lvert t\rvert \geq \mu \\ \frac{r=s-(\frac{2\mu}{\pi}\cdot\sin(\frac{\pi}{2\mu}+\frac{3\pi}{2})+\mu)}{2}, otherwise \end{cases}$ where t=r-s, default setting: MU = 0
normal(MEAN,STDDEV)	none	generates a random number with normal distribution with mean MEAN and standard deviation STDDEV, see MathWorld

© 2022 Prof. Bruce McCarl

pi	any	value of π = 3,141593...
poly(x,A0,A1,A2 [,A3,A4,...])	NLP	computes a polynomial over scalar x, result = A0+A1*x+A2*x^2..., this has a maximum of 6 arguments, default setting: A3,A4,.. = 0
power(x,Y)	NLP	returns x^Y where Y must be an integer
randBinomial(N,P)	none	generates a random number with binomial distribution where n is the number of trials and p the probability of success for each trial, see MathWorld
randLinear (LOW,SLOPE,HIGH)	none	generates a random number between LOW and HIGH with linear distribution, SLOPE must be greater than 2/(HIGH-LOW),
randTriangle (LOW,MID,HIGH)	none	generates a random number between LOW and HIGH with triangular distribution, MID is the most probable number, see MathWorld
rPower(x,y)	NLP	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'.
sigmoid(x)	NLP	sigmoid function as discussed in MathWorld
sign(x)	DNLP	sign of x, returns 1 if x > 0, -1 if x < 0 and 0 if x = 0
signPower(x,Y)	NLP	signed power, returns x^Y if x≥0 and -1*abs(x)^Y if x<0
sin(x)	NLP	returns the sine of the argument x where x must be in radians, see MathWorld
sinh(x)	NLP	returns the hyperbolic sine of x where x must be in radians, see MathWorld
slexp(x[,SP])	NLP	smooth (linear) exponential function, SP means smoothing parameter, default setting: SP = 150
sllog10(x[,SP])	NLP	smooth (linear) logarithm base 10, SP means smoothing parameter, default setting: SP = 10^-150
slrec(x[,SP])	NLP	smooth (linear) reciprocal, SP means smoothing parameter, default setting: SP = 10^-10

© 2022 Prof. Bruce McCarl

sqexp(x[,SP])	NLP	smooth (quadratic) exponential funtion, SP means smoothing parameter, default setting: SP = 150
sqlog10(x[,SP])	NLP	smooth (quadratic) logarithm base 10, SP means smoothing parameter, default setting: SP = 10^-150
sqrec(x[,SP])	NLP	smooth (quadratic) reciprocal, SP means smoothing parameter, default setting: SP = 10^-10
tan(x)	NLP	returns the tangent of the argument x where x must be in radians, see MathWorld
tanh(x)	NLP	returns the hyperbolic tangent of x where x must be in radians, see MathWorld
trunc(x)	DNLP	truncation, removes decimals from x
uniform(LOW,HIGH)	none	generates a random number between LOW and HIGH with uniform distribution, see MathWorld
uniformInt(LOW,HIGH)	none	generates an integer random number between LOW and HIGH with uniform distribution, see MathWorld
vcPower(x,Y)	NLP	returns x^Y for x≥0

7.3.5.2.1.3 Logical Functions

In the following table 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.

All arguments are in lower case which indicates that an endogenous variable is allowed.

Function	**Endogenous Classification**	**Description**
bool_and(x,y)	DNLP	boolean and: returns 0 if x = 0 or y = 0, else returns 1, another possible command is 'x and y'
bool_eqv(x,y)	DNLP	boolean equivalence: returns 0 if exactly one argument is 0, else returns 1,

© 2022 Prof. Bruce McCarl

		another possible command is 'x eqv y'
bool_imp(x,y)	DNLP	boolean implication: returns 1 if x = 0 or y ≠ 0, else returns 0, another possible command is 'x imp y'
bool_not(x)	DNLP	boolean not: returns 1 if x = 0, else returns 0, another possible command is 'not x'
bool_or(x,y)	DNLP	boolean or: returns 0 if x = y = 0, else returns 1, another possible command is 'x or y'
bool_xor(x,y)	DNLP	boolean xor: returns 1 if exactly one argument is 0, else returns 0, another possible command is 'x xor y'
ifThen(cond,iftrue,else)	DNLP	first argument contains a condition (e.g. x > y). If the condition is true, the function returns 'iftrue' else it returns 'else'.
rel_eq(x,y)	DNLP	relation equal: returns 1 if x = y, else returns 0, another possible command is 'x eq y'
rel_ge(x,y)	DNLP	relation greater equal: returns 1 if x ≥ y, else returns 0, another possible command is 'x ge y'
rel_gt(x,y)	DNLP	relation greater than: returns 1 if x > y, else returns 0, another possible command is 'x gt y'
rel_le(x,y)	DNLP	relation less equal: returns 1 if x ≤ y, else returns 0, another possible command is 'x le y'
rel_lt(x,y)	DNLP	relation less than: returns 1 if x < y, else returns 0, another possible command is 'x lt y'
rel_ne(x,y)	DNLP	relation not equal: returns 1 if x ≠ y, else returns 0, another possible command is 'x ne y'

7.3.5.2.1.4 Time and Calender functions

GAMS contains a number of functions that may be used to do things involved with times and dates. The fundamental measurement of time in GAMS is the sequential day number since January 1, 1900, where the integer part of this number contains a unique number for each day while the fractional part contains information about hour, minute, second etc. The day numbers are based on the Gregorian calendar.

In the following table the Endogenous Classification (second column) specifies in which models the function can legally appear. In order of least to most restrictive, the choices are any, NLP, DNLP or none. Note that constant arguments are required.

Function	**Endogenous Classification**	**Description**
gday(SDAY)	any	returns Gregorian day from a serial day number date.time, where Jan 1, 1900 is day 1

© 2022 Prof. Bruce McCarl

gdow(SDAY)	any	returns Gregorian day of week from a serial day number date.time, where Jan 1, 1900 is day 1
ghour(SDAY)	any	returns Gregorian hour of day from a serial day number date.time, where Jan 1, 1900 is day 1
gleap(SDAY)	any	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
gmillisec(SDAY)	any	returns Gregorian milli second from a serial day number date.time, where Jan 1, 1900 is day 1
gminute(SDAY)	any	returns Gregorian minute of hour from a serial day number date.time, where Jan 1, 1900 is day 1
gmonth(SDAY)	any	returns Gregorian month from a serial day number date.time, where Jan 1, 1900 is day 1
gsecond(SDAY)	any	returns Gregorian second of minute from a serial day number date.time, where Jan 1, 1900 is day 1
gyear(SDAY)	any	returns Gregorian year from a serial day number date.time, where Jan 1, 1900 is day 1
jdate (YEAR,MONTH,DAY)	any	returns a serial day number, starting with Jan 1, 1900 as day 1
jnow	none	returns the current time as a serial day number, starting with Jan 1, 1900 as day 1
jstart	none	returns the time of the start of the GAMS job as a serial day number, starting with Jan 1, 1900 as day 1
jtime(HOUR,MIN,SEC)	any	returns fraction of a day that corresponds to hour, minute and second

Example:

(mycalendar.gms)

```
todaydate  = jstart;
now    = jnow;
year   = gyear(todaydate);
month  = gmonth (todaydate);
day    = gday   (todaydate);
```

© 2022 Prof. Bruce McCarl

```
hour   = ghour(todaydate);
minute = gminute(todaydate);
second = gsecond(todaydate);
dow    = gdow (todaydate);
leap   = gleap(todaydate);
display todaydate,now, year, month, day, hour, minute, second, dow, leap;

date  = jdate(year,month,day);
time  = jtime(hour,minute,second);
display date,time;

scalar plus200days;
todaydate  = jstart+200;
year   = gyear(todaydate);
month  = gmonth (todaydate);
day    = gday   (todaydate);
display todaydate,year, month, day;
```

7.3.5.2.1.5 GAMS utility and performance functions

A number of functions can be used to withdraw information on GAMS internal matters. These are addressed as

scalar=functionname;

In the following table the Endogenous Classification (second column) specifies in which models the function can legally appear. In order of least to most restrictive, the choices are any, NLP, DNLP or none. Note that constant arguments are required.

Function	**Endogenous Classification**	**Description**
errorLevel	none	error code of the most recently used command
execError	none	number of execution errors, may either be read or assigned to
gamsRelease	none	returns the version number of the current GAMS release, for example 23.8
gamsVersion	none	returns the current gams version, for example 238
handleCollect(HANDLE)	none	tests if the solve of the problem identified by the argument HANDLE is done and if so loads the solution into GAMS. In particular it returns:

© 2022 Prof. Bruce McCarl

		• 0 if the model associated with HANDLE had not yet finished solution or could not be loaded • 1 if the solution has been loaded
handleDelete(HANDLE)	none	deletes the grid computing problem identified by the HANDLE 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.
handleStatus(HANDLE)	none	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 if the model associaed with HANDLE exists but the solution process is incomplete • 2 if the solution process has terminated and the solution is ready for retrieval • 3 if 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.
handleSubmit(HANDLE)	none	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 if 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 if the completion signal could not be removed • 4 if the resubmit procedure could not be found • 5 if the resubmit process could not be started In case of a nonzero return an execution error is triggered.
heapFree	none	allocated memory which is no more in use but not freed yet
heapLimit	none	interrogates the current heap limit (maximum allowable memory

© 2022 Prof. Bruce McCarl

		use) in Mb and allows it to be reset
heapSize	none	returns the current heap size in Mb
jobHandle	none	returns the Process ID (PID) of the last job started
jobKill(PID)	none	sends a kill signal to the running job with Process ID PID; returns 1 if this was succesful, otherwise 0
jobStatus(PID)	none	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
jobTerminate(PID)	none	sends an interrupt signal to the running job with Process ID PID; returns 1 if this was succesful, otherwise 0
licenseLevel	any	returns an indicator of the 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
licenseStatus	any	returns a non zero when a license error is incurred
maxExecError	none	maximum number of execution errors, may either be read or assigned to
ReadyHandle (handleParameter [, maxWait])	none	can be used to wait until a model solution is finished and ready to be collected. This can be used with Async Threads and grid computing. Possible return values are • 0: (one of) the requested job(s) finished solving and ready to load • 1: there is no active job that GAMS needs to wait for • 2: no valid handle was provided • 3: the handle provided was invalid • 4: user specified time-out when using a SolveLink = %solveLink.Async Threads% handle • 5: user specified time-out when using a SolveLink = %solveLink.Async Grid% handle

© 2022 Prof. Bruce McCarl

		• 8: unknown error (should not happen)
sleep(SEC)	none	execution pauses for SEC seconds
timeClose	none	returns the model closing time
timeComp	none	returns the compilation time in seconds
timeElapsed	none	returns the elapsed time in seconds since the start of a GAMS run
timeExec	none	returns the execution time in seconds
timeStart	none	returns the model start time since last restart

function.gms illustrates the use of assorted functions

7.3.5.2.1.6 String manipulation functions: Ord, Ordascii, Ordebcdic

When called with a string the ord command returns the ASCII code number of a character in a position in a string and is referenced as

```
Ord(string,place);
```

or

```
namedscalar=Ord(string,place);
```

where the strings are just the same as in Card above. The place entry is optional defaulting to one if omitted but otherwise identifying the character position within the text string (1 for the first, 2 for the second etc.) to be examined. The alternative command ordascii may be used in place of ord to return ASCII codes while ordebcdic may be used to return EBCDIC codes.

Example:

(string.gms)

```
$oneolcom
set id namedset /i1 i have explanatory text,ifour,a/;
scalar ii;

ii=Card(id.ts);  !! length of symbol test string
display 'length of text string for symbol', ii;

loop(id,
    ii=Card(id.tl)  !! length of label id   (id must be driving)
    display  'length of set element string', ii;
```

© 2022 Prof. Bruce McCarl

```
   ii=Card(id.te)   !! length of label text (id must be driving)
   display 'length of set explanatory text string', ii;
);

   ii= Card('xxx')  !!  length of 'xxx'
   display 'length of string', ii;

parameter rdd abcdefghijklmnopqrstuvwxyz;
scalar j;
for (j=1 to 27 by 1 ,
   io=Ord(rdd.ts, j);        !! char number
   ia=Ordascii(rdd.ts, j);   !! char number
   ie=Ordebcdic(rdd.ts, j);  !! char number
display 'ords of text string for symbol',j, io,ia,ie;);
```

7.3.5.2.1.7 String manipulation functions: Card

GAMS possesses a few string manipulation functions. These are built on top of Ord and Card. When called with a string the Card command returns the number if characters in the string and is addressed as follows

```
Card(string);
```

or

```
namedscalar=card(string);
```

where string may be

itemname.ts	to get the length of the explanatory text associated with the named item itemname
setname.tl	to get the length of the element name associated with the referenced element of the set setname
setname.te	to get the length of the explanatory text associated with the referenced element of the set setname
"atextstring"	to get the length of the text in a quoted text string

7.3.5.2.1.8 Computer Characteristic Functions

There is currently only one function in this group.

The function numCores returns the number of logical cores on the computer being used.

Ths function can be used at compile time using the syntax

$eval myCores numCores

which places the number of cores into the control variable mycores.

© 2022 Prof. Bruce McCarl

An example of its usage is

```
$eval myCores numCores
$eval halfCores ceil(%myCores%/2)
$log Your system has %myCores% logical processors. GAMS will use %halfCores%.
option threads=%halfCores%;
```

At execution time GAMS one can set a scalar to the number of cores using the sytax

```
s=numcores;
```

7.3.5.2.2 Extrinsic Functions

Users can program their own functions and bring them into GAMS for use in a model. In particular functions can be imported from a user created external library into GAMS then used in a GAMS model. Apart from the commands to cause the import to happen and a naming step, the imported functions can be used in the same way as an internal GAMS functions including in model equations.

Function libraries are imported and made available using the $ command `FuncLibIn as follows`:

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

where the

InternalLibName is the name of the function containg library inside the GAMS code.

ExternalLibName is the name that the library has on the disk drive possibly including its file path.

By default GAMS will look for the library in the GAMS system directory.

To access a library that is not located in the GAMS system directory then the ExternalLibName name must include the absolute path of the library's location.

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 using the **Function** syntax :

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

where

```
InternalFuncName is the name of the function inside the
      GAMS program
InternalLibName  is the name of the loaded library of
      functions and needs to match the name specified in
      a $FuncLibIn command
FuncName         is the name of the function within the
```

© 2022 Prof. Bruce McCarl

```
        loaded library
```

Example: (extrinsic.gms)

Here we replace the existing Cosine, sine and pi functions with alternatives.

```
$eolcom //

$set SLASH \
$if %system.filesys% == UNIX $set SLASH /

$FuncLibIn trilib testlib_ml%SLASH%tridclib  // Make the library available.
                                             // trilib is the internal name bei
                                             // tridclib is the external name.
                                             // With no path, GAMS will look fo
                                             // 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;
```

Comments:

GAMS allows extrinsic function libraries that do not have features to return first derivatives and/or

© 2022 Prof. Bruce McCarl

Hessian values. GAMS approximates these using finite differences. There are GAMS options that allow users to influence the numerical derivative calculations as discussed here. This uses option FDDelta to control step size and FDOpt to control derivative and hessian calculation method and scaling. More is in the McCarl Guide and the release notes.

An example without derivatives can be found in library file trilib which implements a Sine function without derivatives leaving their calculation to GAMS.

The listing file will contain a description of all library's 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
```

There are a few libraries which come ready to use with the GAMS system and also serve as coding examples.

- FITPACK-a collection of FORTRAN programs for curve and surface fitting with splines and tensor product splines developed by P. Dierckx as discussed here at http://nalag.cs.kuleuven.be/research/topics/fitpack.shtml .This does one and two dimensional spline interpolation. An example appears in the GAMS Test Library model fitlib01 the function data needs to be stored in a GDX file fit.gdx which contained a three dimensional parameter fitdata to which the fit will be applied. 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 contain w (weight), x (x-value), y (y-value) or z (z-value).
- Piecewise Polynomial Library - pwpcclib: 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:
- Stochastic Library - stodclib: which provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain distributions. The included distributions are Beta, Cauchy, ChiSquare, F, Gumbel, Inverse Gaussian, Laplace, Logistic , Log Normal, Normal, Pareto, Rayleigh, Student's t, Triangular, Uniform and Weibull plus the discrete distributions Binomial, Geometric, Hypergeometric, Logarithmic, Negative Binomial, Poisson and Uniform Integer. For each there is the capability to generates a random number, a probability density function, a cumulative distribution function and an inverse cumulative distribution function. These can be used in model equations passed to solvers as discussed here or in generating random numbers as discussed here.
- LINDO Sampling Library lsadclib which provides samples of random numbers for certain distributions. It will generate correlated random variables for continuous distributions Beta, Cauchy, Chi-Squared, Exponential, F, Gamma, Gumbel, Laplace, Logistic, Log Normal, Normal, Pareto, Student's t, Triangular , Uniform and Weibull. Also for discrete

© 2022 Prof. Bruce McCarl

distributions Binomial, Hypergeometric , Logarithmic, Negative Binomial and Poisson.These can be used in model equations passed to solvers as discussed here or in generating random numbers as discussed here. A LINDO license is needed to employ these.

- A library that yields PDFs and CDFs for the univariate, bivariate and trivariate normal distributions - cppcclib or cpp is included. This library can automatically compute exact first and second derivatives for the function values being computed. The testlib models cpplib01 and cpplib02 use this.
- A generic approach to interfacing ones own library using trigonometry examples . These are embedded in the examples tricclib, tridclib, trifclib: which have associated source code written in C, Delphi and Fortran.
- One may alternatively use the GAMS =x= features.

More details on the preprogrammed functions appear in the GAMS User's Guide (see appendix J) and here. Further instructions for connecting a specialized library to GAMS can be gained by contacting support@gams.com

In addition, the following examples from the GAMS Test Library can be referred to:

- `stolib01`: Uses the stochastic library which comes with the GAMS system
- `trilib01`: Uses trigonometric function from a library written in C, comes with C source code
- `trilib02`: Uses trigonometric function from a library written in Delphi, comes with Delphi source code
- `trilib03`: Uses trigonometric function from a library written in Fortran, comes with Fortran source code
- `pwplib01`: Uses a library for piecewise polynomial functions which comes with the GAMS system
- `fitlib01`: Uses FITPACK from P. Dierckx which was packaged in a way that it works with this facility and comes with the GAMS system

7.3.5.2.2.1 Probability Distribution Function use in models

GAMS has the ability to include probability density functions and cumulative density functions plus inverse probabilities in models (only the continuous ones) and calculations via use of some provided extrinsic libraries. The continuous distributions across all of these are Beta, Cauchy, ChiSquare, Exponential, F, Gamma, Gumbel, Inverse Gaussian, Laplace, Logistic , Log Normal, Normal,

© 2022 Prof. Bruce McCarl

Pareto, Rayleigh, Student's T, Triangular, Uniform and Weibull plus the discrete distributions Binomial, Geometric, Hypergeometric, Logarithmic, Negative Binomial, Poisson and Uniform Integer.

This involves use of the extrinsic library stodclib. When using these one prefixes the distribution name with

PDF if the probability of a point from the probability density function is needed

CDF if the cumulative probability up to a point from the cumulative density function is needed

ICDF if the value associated with a particular cumulative probability is needed

In addition when prefixing with PDF or CDF one uses a first argument which is the value of the point to be associated with the probability. When using ICDF one uses the probability as the first argument.

A list of the continuous distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution follows

Beta(ALPHA,BETA)	Beta distribution with parameters ALPHA and BETA
Cauchy(MEDIAN,HALFWIDTH)	Cauchy distribution with parameters MEDIAN and HALFWIDTH
ChiSquare(DF)	Chi-squared distribution with the parameter degrees of freedom DF
Exponential(LAMBDA)	Exponential distribution with rate of change parameter LAMBDA
F(DF1,DF2)	F-distribution with parameters for degrees of freedom DF1 and DF2
Gamma(ALPHA, THETA)	Gamma distribution with parameters ALPHA and THETA
Gumbel(ALPHA,BETA)	Gumbel distribution with parameters ALPHA and BETA
InvGaussian(MU,LAMBDA)	Inverse Gaussian distribution with parameters MU and LAMBDA
Laplace(MU,BETA)	Laplace distribution with parameters MU and BETA
Logistic(MU,BETA)	Logistic distribution with parameters MU and BETA
LogNormal(MU,SIGMA)	Log Normal distribution with parameters MU and SIGMA

© 2022 Prof. Bruce McCarl

Normal(MEAN,STD DEV)	Normal distribution with parameters MEAN and STD DEV
Pareto(K,ALPHA)	Pareto distribution with parameters K which gives the min value of the input item and ALPHA the shape parameter
Rayleigh(SIGMA)	Rayleigh distribution with parameter SIGMA
StudentT(DF)	Student's t-distribution with parameter degrees of freedom DF
Triangular(LOW,MODE,HIGH)	Triangular distribution with parameters telling it falls between LOW and HIGH with MODE being the most probable number
Uniform(LOW,HIGH)	Uniform distribution with parameters telling it falls between LOW and HIGH
Weibull(ALPHA,BETA)	Weibull distribution with parameters ALPHA and BETA

These are used first by activating the functions and giving them a local name. For a use involving a mixture of pdfs, cdfs and icdf for the normal, beta, Cauchy and lognormal distributions this is as follows (extrinsicstoc.gms).

```
$funclibin stolib stodclib
function cdfnorm     /stolib.cdfnormal    /
         icdfnorm    /stolib.icdfnormal   /
         pdfnorm     /stolib.pdfnormal    /
         cdfbeta     /stolib.cdfbeta      /
         icdfbeta    /stolib.icdfbeta     /
         cdfcauchy   /stolib.cdfcauchy    /
         icdfcauchy  /stolib.icdfcauchy   /
         cdflognorm  /stolib.cdflognormal /;
```

Where for example the function **CDFNORMAL** that gives the probability up to a specific point in a normal distribution with a given mean and standard deviation is given the local name **cdfnorm**. In turn statements like

```
normalx(i,"cdf")=cdfnorm(normalx(i,"xval"),5,2);
```

where the function is evaluated giving the cumulative probability of the normal distribution from minus infinity to the specified point **normalx(i,"xval")** in a normal distribution with a mean of 5 and a standard deviation of 2.

Examples involving alternative distributions are given below where the first case shows use in **a model equation** and the rest in calculations (extrinsicstoc.gms).

© 2022 Prof. Bruce McCarl

```
totcost.. totalcost=e=costtostock*inventory
        +shortfallcost
        *sum(k,(inventory+del(k)-inventory)
        *(pdfnorm((inventory+del
(k)),meandemand,stddevdemand)));

lognormalx(i,"cdf")
   =cdflognorm(lognormalx
(i,"xval"),lognorm_m,lognorm_s);
normalx(i,"cdf")=cdfnorm(normalx(i,"xval"),10,2);
xvals(i,"val")=icdfnorm(xvals
(i,"prob"),meandemand,stddevdemand);
shouldbepoint5=cdfbeta(xmedian,alpha,beta);
xmedian=icdfcauchy(0.5,median,halfwidth);
x=cdfcauchy(xmedian,median,halfwidth);
```

Note in this case a statement like **icdfcauchy(0.5,median,**halfwidth**)** finds the x value from a Cauchy distribution with parameters median and halfwidth that has a **cumulative probability of 50%** of the observations below it. Also pdfnorm(inventory,meandemand,stddevdemand) returns the probability of the point inventory from the normal distribution with a mean of meandemand and a standard deviation of stddevdemand**.**

When these used in model equations the model needs to be of the type DNLP. Also note the GLOBAL solvers cannot deal with models that contain such functions.

One can also use discrete distributions but only in calculations. A list of the discrete distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution follows

The discrete distributions included are (Note these cannot be used in model .. equations)

Binomial(N,P)	Binomial distribution with parameters for number of trials N and success probability P in each trial
Geometric(P)	Geometric distribution with parameter giving success probability P in each trial
HyperGeo(TOTAL,GOOD,TRIALS)	Hypergeometric distribution with parameters giving total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS
Logarithmic(THETA)	Logarithmic distribution with parameter THETA, also called log-series distribution

© 2022 Prof. Bruce McCarl

NegBinomial(FAILURES,P)	Negative Binomial distribution with parameters for the number of failures until the experiment is stopped FAILURES and the success probability P in each trial. The number generated by PDF and CDF or input to ICDF describes the number of successes before the de?ned number of failures
Poisson(LAMBDA)	Poisson distribution with mean LAMBDA
UniformInt(LOW,HIGH)	Integer Uniform distribution with parameters telling that the distribution falls between LOW and HIGH values.

Again these are referenced with the PDF, CDF and ICDF prefixes. In addition when prefixing with PDF or CDF one uses a first argument which is the value of the point to be associated with the probability. When using ICDF one uses the probability as the first argument.

Use of these is not allowed in model equations and when attempted causes an execution error.

7.3.5.2.2.2 Random Numbers from Functions

Capabilities have been introduced to generate random numbers from a variety of **distributions** again via extrinsic functions. To do this one can use variants of the functions just discussed (those from the Stochastic Library **stocldib**) or ones from the Lindo Sampling Library (**lsadclib) although use of the Lindo ones requires a license for GAMS/Lindo**. (Without a license only a demo version is available which is restricted to the Normal and the Uniform distribution and no more than 10 sample points.)

Using the Stochastic Library stocldib

In terms of the Stochastic Library the usable continuous and discrete distributions are those listed in the tables above. When using that library one again needs to activate the library and give the function a local name. Here to generate random numbers the distribution name is prefixed with a d. Thus, for example, to generate numbers from the Cauchy, Binomial, Normal and LogNormal one would use code like the following (randnumbers.gms)

```
$funclibin stolib stodclib
function randnorm     /stolib.dnormal    /
         randbin      /stolib.dbinomial  /
         randcauchy   /stolib.dcauchy    /
         randlognorm  /stolib.dlognormal /;
set i /i1*i20/
set j /norm,binomial,cauchy,lognorm/
parameter randx(i,j)    numbers from distributions;
```

© 2022 Prof. Bruce McCarl

```
randx(i,"norm")=randnorm(5,2);
randx(i,"binomial")=randbin(10,0.5);
randx(i,"cauchy")=randcauchy(5,1);
randx(i,"lognorm")=randlognorm(1.2,0.3);
display randx;
```

That code first activates the stoclib library then gives the desired functions local names. Subsequently the functions are used in this case generating 20 random numbers for each of the 4 distributions with the desired parameters.

One can also reset the random number seed using a function SetSeed(SEED) as follows

```
function setseedh     /stolib.SetSeed    /;
scalar x;
x=setseedh(99883);
```

where a local function name is again defined and then the seed is set by setting a scalar equal to the function with a user defined value that when this is set to the same value between runs will cause the random numbers to always follow a particular sequence.

Using the Lindo Sampling Library lsadclib

In terms of the Lindo Sampling Library, one can generate random numbers for many of the same distributions with a few additional ones as listed below. In addition one has the capability to introduce correlations and exercise control over the random number process. Note these cannot be used in model equations.

The continuous distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution are listed in the table below. Here note that the parameters here also include

- the sample size (SAMSIZE) telling how many numbers to generate
- an optional parameter (VARRED)that allows one to alter the sampling procedure variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic) with Latin Hyper Square Sampling being the default.

Beta(ALPHA,BETA,SAMSIZE[,VARRED])	Beta distribution with parameters ALPHA and BETA
Cauchy(MEDIAN,HALFWIDTH,SAMSIZE[,VARRED])	Cauchy distribution with parameters MEDIAN and HALFWIDTH

© 2022 Prof. Bruce McCarl

ChiSquare(DF,SAMSIZE[,VARRED])	Chi-squared distribution with the parameter degrees of freedom DF
Exponential(LAMBDA,SAMSIZE[,VARRED])	Exponential distribution with rate of change parameter LAMBDA
F(DF1,DF2,SAMSIZE[,VARRED])	F-distribution with parameters for degrees of freedom DF1 and DF2
Gamma(ALPHA, THETA,SAMSIZE[,VARRED])	Gamma distribution with parameters ALPHA and THETA
Gumbel(ALPHA,BETA,SAMSIZE[,VARRED])	Gumbel distribution with parameters ALPHA and BETA
InvGaussian(MU,LAMBDA,SAMSIZE[,VARRED])	Inverse Gaussian distribution with parameters MU and LAMBDA
Laplace(MU,BETA,SAMSIZE[,VARRED])	Laplace distribution with parameters MU and BETA
Logistic(MU,BETA,SAMSIZE[,VARRED])	Logistic distribution with parameters MU and BETA
LogNormal(MU,SIGMA,SAMSIZE[,VARRED])	Log Normal distribution with parameters MU and SIGMA
Normal(MEAN,STD DEV,SAMSIZE[,VARRED])	Normal distribution with parameters MEAN and STD DEV
Pareto(K,ALPHA,SAMSIZE[,VARRED])	Pareto distribution with parameters K which gives the min value of the input item and ALPHA the shape parameter
Rayleigh(SIGMA,SAMSIZE[,VARRED])	Rayleigh distribution with parameter SIGMA
StudentT(DF,SAMSIZE[,VARRED])	Student's t-distribution with parameter degrees of freedom DF
Triangular(LOW,MODE,HIGH,SAMSIZE[,VARRED])	Triangular distribution with parameters

© 2022 Prof. Bruce McCarl

	telling it falls between LOW and HIGH with MODE being the most probable number
Uniform(LOW,HIGH,SAMSIZE[,VARRED])	Uniform distribution with parameters telling it falls between LOW and HIGH
Weibull(ALPHA,BETA,SAMSIZE[,VARRED])	Weibull distribution with parameters ALPHA and BETA

The discrete distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution are listed in the table below. Here note that the parameters also include

- the sample size (SAMSIZE) telling how many numbers to generate
- an optional parameter (VARRED)that allows one to alter the sampling procedure variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic) with Latin Hyper Square Sampling being the default.

Binomial(N,P,SAMSIZE[,VARRED])	Binomial distribution with parameters for number of trials N and success probability P in each trial
HyperGeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])	Hypergeometric distribution with parameters giving total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS
Logarithmic(THETA,SAMSIZE[,VARRED])	Logarithmic distribution with parameter THETA, also called log-series distribution
NegBinomial(SUCCESS,P,SAMSIZE[,VARRED])	Negative Binomial distribution with parameters for the number of successes (SUCCESS) to be achieved and the probability of

© 2022 Prof. Bruce McCarl

	success in each trial (P). The generated random number describes the number of failures until we reached the specified number of successes. Note that the Lindo sampling library version is equivalent to the use of the one from the stochastic library when the probability there is 1-P here.
Poisson(LAMBDA)	Poisson distribution with mean LAMBDA

Table J.6: LINDO sampling functions

When using the Lindo Sampling Library one again needs to activate the library and give the functions to be used a local name where here the distribution name is prefixed with SampleLS. For example, to generate 25 numbers from the Cauchy, Binomial, Normal and LogNormal using the default variance reduction method, one would use code like the following (randnumbers.gms). In that code we first activate the library and give the items local names then in the Lindo Sampling case must set a scalar pointer to the function with the distribution arguments and sample size. This returns a pointer to a location identifying where the random numbers are stored. In turn then we use a loop statement to load in the random numbers using a local version of another function getSampleValues referencing the scalar pointer that sequentially returns each of the values in the sample when called. In turn these need to be stored into a GAMS parameter (randnumbers.gms).

```
$funclibin lsalib lsadclib
function normalSample    /lsalib.SampleLSnormal /
         lognormalSample /lsalib.SampleLSlognormal /
         cauchySample    /lsalib.SampleLScauchy /
         binomialSample  /lsalib.SampleLSbinomial /
         getSampleVal    /lsalib.getSampleValues /;

scalar d,h,k,k1,k2;
set is /value01*value12/;
parameter sv(is,j);
k = normalSample(5,2,12);
loop(is, sv(is,"normal") = getSampleVal(k); );
h = lognormalSample(1.2,0.3,12);
loop(is, sv(is,"lognormal") = getSampleVal(h));
k1 = binomialSample(10,0.5,12);
loop(is, sv(is,"binomial") = getSampleVal(k1));
k2 = cauchySample(5,1,12);
loop(is, sv(is,"binomial") = getSampleVal(k2));
```

© 2022 Prof. Bruce McCarl

One can also cause items to be correlated using the functions **setCorrelation and induceCorrelation.** The parameters for this are

setCorrelation(POINTER1,POINTER2,COR)	Causes subsequent calls involving the samples identifies by POINTER1 and POINTER2 to have the correlation COR when next loaded using the getSampleVal function.
induceCorrelation(TYPE)	This controls the type of correlation that is being specified where TYPE is 0 for Pearson, 1 for Kendall or 2 for Spearman. It must be used after setcorrelation.

Coding to make this happen follows. Again we activate the procedures and give them local names then generate the random numbers for two same sized samples. Afterwards we reference set the correlation with arguments of the pointers to the two series then the correlation coefficient and its form. Finally the loop statement is used again to draw out the values.

```
function setCor          /lsalib.setCorrelation /
         indCor          /lsalib.induceCorrelation /;
k = normalSample(5,2,12);
h = lognormalSample(1.2,0.3,12);
d=setCor(h,k,-1);
d=indCor(1);
loop(is,sv2(is,"aftercorr","normal") = getSampleVal
(k)););
loop(is,sv2(is,"aftercorr","lognormal") = getSampleVal
(h)););
```

There are also function alternatives to set the random number seed and the type of random number generation process

setSeed(SEED)	the SEED arguments resets the random number generator according to a user defined value
setRNG(RNG)	The parameter RNG identifies the random number generator to use with possible values of -1 (free), 0 (system), 1 (lindo1), 2 (lindo2), 3 (lin1), 4 (mult1), 5 (mult2) and 6 (mersenne).

© 2022 Prof. Bruce McCarl

Using the Stochastic Library CPP (cppcclib)

A library variously called **CPP** and **cppcclib** contains provisions to generate probability density function values and cumulative probability density function values from univariate and multivariate normal distributions. The distributions are assumed to have a mean of zero and a variance of one.

Function	Description	Solver requirement
cdfUVN(x)	cdf value giving cumulative probability up to the point x (ie x standard deviations from the mean) from univariate normal with mean zero and variance 1	Nonlinear requiring NLP
pdfBVN(x,y,r)	Pdf value giving the bivariate normal probability for the point x, y (ie x and y standard deviations from the mean) where there is correlation r between x and y. This is from a bivariate normal with mean at zero for x and y and variance of 1 for all	Nonlinear requiring NLP
cdfBVN(x,y,r)	Cdf value giving the bivariate normal cumulative probability up to the point x, y (ie x and y standard deviations from the mean) where there is correlation r between x and y. This is from a normal with mean at zero for x and y and variance of 1 for all	Nonlinear requiring NLP
pdfTVN (x,y,z,r21,r31,r32)	Pdf value giving the trivariate normal probability for the point x, y, z (ie x , y and z standard deviations from the mean) where there is correlation r21 between x and y, r31 between x and z and r32 between y and z. This is from a trivariate normal with mean at zero for x, y, z and variance of 1 for all	Nonlinear requiring NLP
cdfTVN (x,y,z,r21,r31,r32)	cdf value giving the trivariate normal probability up to the point x, y, z (ie x , y and z standard deviations from the mean) where there is correlation r21 between x and y, r31 between x and z and r32 between y and z. This is from a trivariate normal with mean at zero for x, y, z and variance of 1 for all	Nonlinear requiring NLP

More on this is found in the GAMS user guide Appendix J on intrinsic functions under the topic

CPP Library

© 2022 Prof. Bruce McCarl

7.3.5.2.2.3 Defining an Extrinsic Function

Some users may wish to define their own custom extrinsic library. In doing this several steps are recommended.

- Read appendix J of the GAMS user guide to get some idea of basic functionality.
- Identify a function that you wish to be able to use from within GAMS and GAMS solves (excepting the global ones that cannot use extrinsic functions). In doing this insure that you cannot implement the function as a macro or batinclude file as those implementations would be much simpler. One may alternatively use the GAMS =x= features.
- If you want to use the function in a model sent to a solver (ie within an equation) with endogenous variables as arguments, make sure you know how to compute first and second derivatives with respect to all the variables.
- Choose a programming language. Obvious candidates for a programming language are C, C ++, Delphi, or Java, as GAMS makes examples available in these languages. These are available in association with the trigonometric library (trifclib) and the GAMS test library contains examples and source codes. If you download using the IDE library manager note that the manager will place the source zip code including a make file in your project file area.
- Using the demonstration models from the test library, make sure you can exercise the entire process of building and testing the extrinsic function shared library as now done for trifclib in the language of your choice. For example, if you're using C++, look at the testlib model trilib01 or cpplib01. If you're using Fortran, start with trilib03.
- Assemble and test some code that evaluates your function and its derivatives. Note when called the fist parameter tells whether to return the function evaluation, its first derivative or its second derivative.
- Integrate your code into the example code from GAMS using the source files provided for the demo library as a template. This will involve modifying the .spec file to describe the function you are creating and creating the API files based on the .spec file, and corresponding edits to the source code actually implementing the function.
- Optionally, if this seems like a big step, start small. Implement a simple function (e.g. sqr(sum of x)) for which you know the gradients and Hessians, and get the interface to do this right. Then substitute the real function of interest.
- Do some rigorous, extensive testing of your function. Many examples exist in testlib for doing this, e.g. cpplib01 * cpplib05. The tests might involve pre-computing input arguments and known function values in another language and storing these values in a GDX file for comparison. Consider testing very small or very large values, and testing how errors in the function will be handled.
- The extrinsic functions are limited to 20 scalar arguments and return a scalar value.

This requires substantial programming ability and one may need to get tips from

© 2022 Prof. Bruce McCarl

support@gams.com .

7.3.5.3 Special values

GAMS uses special symbols to handle missing data, the results of undefined operations, small nonzero results, and the representation of bounds that solver systems regard as 'infinite.' The special symbols are listed below with a brief explanation of the meaning of each.

Inf, -Inf
Na
Eps
Undf
Yes/No

7.3.5.3.1 Inf, -Inf

These values are plus and minus infinity. Inf is a very large positive number. –Inf is a very large in absolute value negative number. They may be used in an assignment as follows (**specval.gms**)

```
z$y=x/y;
if(y =0,z=inf);
z$y=-x/y;
if(y =0,z=-inf);
```

through such mechanisms one can reflect cases where infinite results arise.

7.3.5.3.2 Na

This value indicates a result is not available. Note any operation that uses the value NA will produce the result NA. This may be used in an assignment as follows (**specval.gms**)

```
z$y=x/y;
if(y =0,z=na);
```

7.3.5.3.3 Eps

This value indicates a result is very close to zero, but is different from zero. This may be used in an assignment as follows (**specval.gms**)

```
z$y=x/10000000000000;
if(y < 0.00001,z=EPS);
```

7.3.5.3.4 Undf

This value indicates a result is undefined. A user may not use this in an assignment unless the $onundf command has been used. GAMS will assign it in undefined cases such as the one below.

```
z=x/0;
```

© 2022 Prof. Bruce McCarl

```
display z;
```

7.3.5.3.5 Yes/No

These values are used to assign elements to subsets as discussed in the sets chapter in the element defined by calculation section.

7.3.5.4 Model and optimal solution items

Information may be defined relative to variables, equations and models that pass back and forth between GAMS and solvers. These items can be computed or used in computations as discussed below.

7.3.5.4.1 Attributes of variables and equations

Variables and equations have attributes that are passed to solvers, are passed back from solvers or are used in scaling the model. These may be used in computations or have values assigned to them through computations.

L
M
Lo
Up
Fx
Scale
Prior

7.3.5.4.1.1 L

Variables and equations have what is known as a level value that is the current solution value for that variable or equation. The level value for a variable is the current solution value after a solution and the starting point for that variable before solution plus it helps provide a basis for the model. The level value for an equation is sum of the endogenous terms in the equation evaluated at the current level value for the variables after a solution and in general is not defined before solution. The attribute is addressed using .L as follows

```
Variable level is variablename.L(setdependency)
Equation level is equationname.L(setdependency)
```

These are commonly used in calculations to do one of three things

- Variable levels are set at non zero levels pre solution to provide a starting point that can be very important in NLP problems.
- Variable levels are set at nonzero values pre solution to provide an advanced basis.
- Variable and equation levels are used post optimality in report writing (Note in this circumstance $Ondotl may be used to automatically add the .l items to variable names appearing on the right hand side of calculations).

© 2022 Prof. Bruce McCarl

Example:

(varmodatt.gms)

```
X.l(j)=1;
Move.l(source,sink)=historicmove(source,sink);
solve transport using lp minimizing cost;
Totalcost=sum((source,sink ),
     costtomove(source,sink )*Move.l(source,sink ));
```

Note:

- The level values receives new values after a solve statement is executed.
- One can use $ondotl to add .l to variable names automatically.

7.3.5.4.1.2 M

Variables and equations have what is known as a marginal value that is the current shadow price or reduced cost for that item. The marginal value for a variable is the current reduced cost after a solution plus it helps provide a basis for the model. The marginal value for an equation is the shadow price for the equation and in general is not defined before solution but if present helps provide a basis for the model. The attribute is addressed using .M as follows

```
Variable marginal is variablename.M(setdependency)
Equation marginal is equationname.M(setdependency)
```

These are commonly used in calculations to do one of two things

- Variable marginals are set at nonzero values pre solution to provide an advanced basis.
- Variable and equation marginals are used post optimality in report writing.

```
incoming(destinaton,"Marg Cost of","meeting needs"," ","Total")  =
      demandbal.m(destinaton);
```

Example:

(varmodatt.gms)

```
sinkdemand.M(sink)=1;
sourcesupply.M(source)=1;
solve transport using lp minimizing cost;
shadowprices("demand",sink)=sinkdemand.M(sink);
shadowprices("supply",source)=sourcesupply.M(source);
```

Note:

- The marginal values receives new data after a solve statement is executed.

© 2022 Prof. Bruce McCarl

7.3.5.4.1.3 Lo

Variables and equations have a lower bound. The lower bound for a variable is the smallest value that the variable can take on. The lower bound for an equation is the right hand side for a greater than or equal to and equality constraint. It is minus infinity for a less than or equal to constraint. The attribute is addressed using .Lo as follows

```
Variable lower bound is variablename.lo(setdependency)
Equation lower bound is equationname.lo(setdependency)
```

These are commonly used in calculations pre solution to specify or revise the bound.

Example:

(varmodatt.gms)

```
Move.lo(source,sink)=0.1;
solve transport using lp minimizing cost;
display sinkdemand.Lo;
```

Notes:

- The lower bound is also changed when the fx attribute for a variable is set.
- The lower bound defaults to 0 for positive variables or –inf for other unrestricted or negative variables.
- The lower bound defaults to -inf for less than or equal to equations.

7.3.5.4.1.4 .range

Variables and equations have an upper and lower bounds. The range for a variable is the difference between these bounds.

Variable range is variablename.range (setdependency)
Equation range is equationname.range (setdependency)

These can be used in calculations to see if a variable is fixed revise the bound.

Example:

(varmodatt.gms)

```
display move.range;
```

Notes:

- The range becomes zero when the .fx attribute for a variable is set.

© 2022 Prof. Bruce McCarl

- The .range is defined as x.range=x.up−x.lo.
- This provides a convenient way to see if a variable is fixed

7.3.5.4.1.5 Up

Variables and equations have an upper bound. The upper bound for a variable is the largest value a variable can take on. The upper bound for an equation is the right hand side for a less than or equal to and an equality constraint. It is plus infinity for a greater than or equal to constraint. The attribute is addressed using .Up as follows

```
Variable upper bound is variablename.up (setdependency)
Equation upper bound is equationname. up (setdependency)
```

These are commonly used in calculations pre solution to specify or revise the bound.

Example:

(varmodatt.gms)

```
Move.up(source,sink)=1000.1;
solve transport using lp minimizing cost;
display sinkdemand.up;
```

Notes:

- The upper bound is also changed when the .fx attribute for a variable is set.
- The upper bound defaults to +inf for positive and unrestricted variables and 0 for negative variables.
- The upper bound defaults to +inf for greater than or equal to equations.

7.3.5.4.1.6 Fx

Variables can be fixed at a value. In turn GAMS sets the lower and upper bound to that value. The attribute is addressed using .Fx as follows

```
Variable fixed level is variablename.fx (setdependency)
```

These are commonly used in calculations pre solution to specify or revise the value.

Example:

(varmodatt.gms)

```
Move.fx("boston","seattle")=1.;
solve transport using lp minimizing cost;
display move.lo,move.fx;
```

© 2022 Prof. Bruce McCarl

Notes:

- The upper and lower bound are changed when the Fx attribute for a variable is set to the value of the Fx attribute.
- Fixed variables can subsequently be freed by changing the lower .lo and upper .up bounds.
- Fx attributes are not defined for equations.
- One cannot use the Fx attribute in the expression in an equation or on the right hand side of a replacement statement.

7.3.5.4.1.7 Scale

Variables and equations have a scale attribute. The scale attribute for a variable is a number that all coefficients associated with that variable are multiplied by. The scale attribute for an equation is a number that all coefficients associated with that equation are divided by. The attribute is addressed using .Scale as follows

```
Variable scale factor is variablename.scale(setdependency)
Equation scale factor is equationname.scale(setdependency)
```

These are commonly used in calculations pre solution to specify or revise the scaling factors.

Example:

(varmodatt.gms)

```
Transport.scaleopt=1;
Move.scale("boston","seattle")=10.;
```

Notes:

- The scale factors only work if the modelname.scaleopt is set to a non zero value.
- Scaling is discussed in the Scaling GAMS Models chapter.
- Scaling factors default to one.

7.3.5.4.1.8 Prior

Variables in mixed integer programs can have a priority attribute. The user can use this parameter to specify an order for picking variables to branch on during a branch and bound search for MIP model solution. Without priorities, the MIP algorithm will determine the variable most suitable to branch on. One must also tell GAMS statement to use priorities by entering

```
modelname.prioropt = 1 ;
```

where modelname 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 attribute sets the priorities of the individual

© 2022 Prof. Bruce McCarl

variables. Priorities can be set to any real value. The default value is 1.0. As a general rule of thumb, the most important variables should be given the highest priority which implies they should have the lowest nonzero values of the .prior attribute. The attribute is addressed using .Prior as follows

```
Variable priority is variablename.prior(setdependency)
```

These are commonly used in calculations pre solution to specify or revise the priority factors.

Example:

```
Move.prior("boston","seattle")=1.;
```

Notes:

- The prior factors only work with integer variables.
- The prior factors require that modelname.prioropt be set to a non zero value.

7.3.5.4.2 Attributes of models

Model attributes contain information relative to model solution procedures and results passed back and forth between GAMS and solvers.

There are two fundamental types of model attributes:

- Attributes mainly used before a solve statement to provide information to GAMS or the solver link.
- Attributes mainly used after a solve statement to retrieve information about the last solve.

For details see Model Attributes

7.3.6 Including conditionals

Calculations can include conditionals as extensively covered in the conditionals chapter. One very important distinction needs to be bought out regarding conditional placement and equation result. This involves the distinction between a right and left hand side conditional.

Right and left hand side conditionals

7.3.6.1 Right and left hand side conditionals

Conditionals may be used on either the right or left hand side of assignment statements but the operation is very different.

- A conditional on the left-hand side specifies the condition under which a statement is executed. Thus, no assignment is made unless the conditional is satisfied.
- A conditional on the right-hand side specifies the condition under which a term is computed. Thus, the assignment is always made but the term is zero unless the conditional is satisfied.

This means that with a conditional on the left hand side of an assignment the previous contents of the parameter will remain unchanged for conditionals that are not satisfied.

© 2022 Prof. Bruce McCarl

For example in the GAMS code below (**leftright.gms**)

```
Y=2;
Z=2;
X=0;
Y$X=4;
Z=4$X;
```

Y will end with a value of 2 but Z will equal zero since the Y calculation with the left hand side conditional will not be executed since X=0. But the Z calculation is always done and the right hand side conditional 4$X term will be zero since X is zero.

Analogous conditions hold for models where

- A conditional on the left-hand side of the .. specifies the condition under which an equation is defined. Thus, the constraint equation is not present unless the conditional is satisfied.
- A conditional on the right-hand side after the .. specifies the condition under which a term is computed in the equation. Thus, the equation is defined but the term is zero unless the conditional is satisfied.

This means that with a left hand side conditional the equation is not entered into the model.

- For example in the GAMS code below (**leftright.gms**)

```
C1$x..    YY=e=4;
C2..      YY=e=4$X;
```

YY will not be constrained by C1 since it has a left hand side conditional and will not be defined as an equation since X=0. But the C2 equation is always defined and the YY will be set equal to zero since the right hand side conditional 4$X term becomes zero since X is zero.

7.4 Improving Output via Report Writing

Generally, the GAMS model solution output is not adequate for conveying solution information to the modeler or associated decision-makers. Report writing coupled with item displays can help. However that is sometimes is not in a satisfactory format and further efforts may be necessary. Here I discuss these topics.

Adding report writing
Using displays
Formatting pages and lines
Output via put commands
Reordering set order in output
Preprogrammed table making utility: Gams2tbl
Output to other programs
Obtaining graphical output
Sorting output

© 2022 Prof. Bruce McCarl

7.4.1 Adding report writing

GAMS permits one to do calculations using solution information to improve the information content of the output and display the calculated items. This exercise is commonly called report writing.

The basic report writing approach involves three phases:

- Design of a tabular format for presentation of model study results.
- Calculation of entries for which presentation based on a mixture of raw and model solution data.
- Display of that information.

I will not cover the first of these three points as their use depends on the study but will cover the second two.

Basics of solution based report writing calculations
Adding a report
Notes on indefinite sets in parameter statements

7.4.1.1 Basics of solution based report writing calculations

Information relative to the variable, equation and model solution is passed to GAMS from solvers. These information are used in report writing computations (see the Calculating Items chapter for further discussion).

Variables and equations have attributes that contain the optimal level and marginal values for them in the current solution.

- The level value attribute for a variable is the current optimal value in the solution. The attribute is addressed using .L as follows

```
Variablename.L(setdependency)
```

  This may also be done implicitly using `$Ondotl`.

- The level value attribute for an equation is the endogenous terms in the equation evaluated at the current solution value for the variables after a solution. The attribute is addressed using .L as follows:

```
Equationname.L(setdependency)
```

- The marginal attribute for a variable is the current reduced cost to force a variable in the solution. The attribute is addressed using .M as follows

```
Variablename.M(setdependency)
```

- The marginal value attribute for an equation is the dual value or shadow price for the equations after a solution. The attribute is addressed using .M as follows:

© 2022 Prof. Bruce McCarl

```
Equationname.M(setdependency)
```

The numerical values of these parameters are generally redefined every time a solve is executed. In general, these items can be used in calculations just like any other parameter once a solve has been completed.

Note model solution attributes like modelname.modelstat can also be used to indicate whether an optimal solution was attained as discussed in the Model Attributes chapter.

7.4.1.2 Adding a report

Now suppose I add a report. Lets do this in a simple transportation model. First, let me briefly design it. Lets suppose for each destination I wish to know where the goods came from, how much the marginal cost of meeting demand is and what is the total shipment cost to that location. I will do this in a rather quick and dirty fashion using a five dimensional parameter defined over unspecified set elements employing the universal set concept discussed below or in the Sets chapter.

I do this in the transportation model **calcoutp.gms** by computing a parameter called incoming. There I

- Define a parameter called incoming as a 5 dimensional entity with unspecified set elements.
- Place the incoming shipment levels (transport.l(source,destinaton)) into that parameter with appropriate labeling.
- Place the marginals from the demand rows **demandbal.m(destinaton)** into that parameter with appropriate labeling.
- Compute a total cost of shipping by multiplying per unit cost times the volume shipped and summing over all incoming routes.
- Compute total shipments from a place and **total cost of shipping** adding up the data already in the incoming parameter with appropriate labeling.
- **Display the result** using the formatting options described below.

The added code to do this is

```
parameter incoming(*,*,*,*,*) incoming shipment report;
incoming(destinaton,"shipments","in cases","from",source)
      =transport.l(source,destinaton);
incoming(destinaton,"Marg Cost of","meeting needs"," ","Total")
      =demandbal.m(destinaton);
incoming(destinaton,"Cost of","shipping"," ","total")      =sum(source,
trancost(source,destinaton)*transport.l(source,destinaton));
incoming("total","shipments","in cases","from",source)
  =sum(destinaton,
     incoming(destinaton,"shipments","in cases","from",source));
incoming("Total","Cost of","shipping"," ","total")
  =sum(destinaton,
     incoming(destinaton,"Cost of","shipping"," ","total"));
option incoming:0:3:2;display incoming;
```

© 2022 Prof. Bruce McCarl

This results in the following report with the results color coded to the originating statement.

```
----     79 PARAMETER INCOMING         incoming  shipment report

                                             from         from
                                          Seattle    San Diego      Total
New York.Shipments   .in cases                 50          275
New York.Cost of     .shipping                                      81250
New York.Marg Cost of.meeting needs                                   250
Chicago .Shipments   .in cases                300
Chicago .Cost of     .shipping                                      53400
Chicago .Marg Cost of.meeting needs                                   178
Topeka  .Shipments   .in cases                          275
Topeka  .Cost of     .shipping                                      41525
Topeka  .Marg Cost of.meeting needs                                   151
Total   .Shipments   .in cases                350          550
Total   .Cost of     .shipping                                     176175
```

7.4.1.3 Notes on indefinite sets in parameter statements

Parameter statements do not always have to have definitive set assignments. If one enters an * in an index position, then anything at all can be placed in that position. In above example, I use this allowing the inclusion of both existing sets and other items in these positions. For example, in the first position of the incoming parameter I am using both the set destination and the element text "total".

```
incoming(destinaton,"shipments","in cases","from",source)
incoming("total","shipments","in cases","from",source)
```

- Note one should not use an * in declarations in place of set names when inputting table or parameter data. Only use it for output data and then infrequently.
- This is discussed more fully in the Sets chapter.

Use of named sets in the specifications allows GAMS to check to make sure you're not misspelling anything.

7.4.2 Using displays

One may display any GAMS parameter, set, variable attribute, equation attribute or model attribute as well as quoted text using the GAMS display statement. Note: Display will not print out items that are zero leaving blanks or skipping items where entire rows or columns are zero. Generally the display is of the format

```
DISPLAY ITEM1,ITEM2,ITEM3;
```

where the items are either

- Quoted strings in single or double quotes such as

```
Display 'here it is', "hello";
```

© 2022 Prof. Bruce McCarl

- Parameter or set names without any referencing to setdependency. Thus in **dispord.gms** while the parameter data is defined over 4 sets

```
parameter data(index1,index2,index3,index4);
```

I simply say

```
display data;
```

- Variable, equation or model attributes with the item name and attribute desired specified

```
Display x.l, x.scale, modelname.solstat;
```

- Multiple items can be listed in a display statement separated by commas.
- When displaying a more than 2 dimensional item one can expand the width of the labels printed out beyond 10 using the command **option dispwidth=number;** where number can range up to 31.

Abort
Controlling displays

7.4.2.1 Abort

The abort command can also be used to display data but once encountered causes the program to stop with an execution error. The command syntax is just like that for a display command with the same syntax excepting the word abort replacing display. Abort usage is illustrated in **abort.gms** and its use is discussed in the Conditionals chapter.

7.4.2.2 Controlling displays

GAMS displays can be enhanced in terms of form, and content.

7.4.2.2.1 Formatting display decimals and layout

Users may not find the GAMS display style consistent with what they want. The GAMS OPTION statement permits one to alter this. In particular, an option statement of the form

```
OPTION THISITEM:DECIMAL:ROWitems:COLUMNitems
```

can modify the display formatting. Use of this option will cause all subsequent displays of the item named THISITEM to follow rules specified by the three numbers following the colons which are

DECIMAL	number of decimal places to be included
ROWitems	number of indices displayed within rows
COLUMNitems	number of indices displayed within columns

© 2022 Prof. Bruce McCarl

For DATA(A,B,C) I could have rowitems:columnitems values of 2:1, 1:2, 0:3 or 1:1. The first (2:1) would have A&B varied in the rows with C defined in the columns. The second (1:2) would have A in the rows with the B&C in the columns. The third (0:3) would list everything in columns. The fourth (1:1) would have a table for each A element that contained B in rows and C in the columns.

Column label width can be expanded beyond the default of 10 using **option dispwidth=n** where n can go up to 31.

Example:

(dispord.gms)

Suppose I have a four-dimensional array, each dimension of which has two elements. An ordinary display of this would yield:

```
INDEX 1 = index11

                    index41     index42
index21.index31       2.000       2.000
index21.index32       2.000       2.000
index22.index31       2.000       2.000
index22.index32       2.000       2.000

INDEX 1 = index12

                    index41     index42
index21.index31       2.000       2.000
index21.index32       2.000       2.000
index22.index31       2.000       2.000
index22.index32       2.000       2.000
```

Use of the option statement could manipulate it into number of different forms including

```
 option data:0:1:3;display data;

----      8 PARAMETER DATA
            index21     index21     index21     index21     index22
            index31     index31     index32     index32     index31
            index41     index42     index41     index42     index41

index11           2           2           2           2           2
index12           2           2           2           2           2

      +     index22     index22     index22
            index31     index32     index32
            index42     index41     index42

index11           2           2           2
index12           2           2           2

 option data:0:3:1;display data;
```

© 2022 Prof. Bruce McCarl

```
                              index41     index42
index11.index21.index31             2           2
index11.index21.index32             2           2
index11.index22.index31             2           2
index11.index22.index32             2           2
index12.index21.index31             2           2
index12.index21.index32             2           2
index12.index22.index31             2           2
index12.index22.index32             2           2
```

```
 option data:0:0:4;display data;
```

```
index11.index21.index31.index41 2,    index11.index21.index31.index42 2
index11.index21.index32.index41 2,    index11.index21.index32.index42 2
index11.index22.index31.index41 2,    index11.index22.index31.index42 2
index11.index22.index32.index41 2,    index11.index22.index32.index42 2
index12.index21.index31.index41 2,    index12.index21.index31.index42 2
index12.index21.index32.index41 2,    index12.index21.index32.index42 2
index12.index22.index31.index41 2,    index12.index22.index31.index42 2
index12.index22.index32.index41 2,    index12.index22.index32.index42 2
```

```
 option data:0:2:2;display data;
```

```
                   index31     index31     index32     index32
                   index41     index42     index41     index42

index11.index21          2           2           2           2
index11.index22          2           2           2           2
index12.index21          2           2           2           2
index12.index22          2           2           2           2
```

Notes:

- If one specifies less dimensions in the option than the number of sets i.e. indicating two by one with a item defined over 4 sets, then the extra dimensions will be used by the earliest sets in the specification and a different table will be output for each set element in those early sets. Thus when the option command

```
option data:0:2:1;display data;
```

specifying placement of 3 sets in the rows and columns is used on the example where the parameter is defined over 4 sets I get

```
INDEX 1          = index11
                    index41     index42
index21.index31           2           2
index21.index32           2           2
index22.index31           2           2
```

© 2022 Prof. Bruce McCarl

```
index22.index32          2          2

INDEX 1          = index12
                    index41    index42
index21.index31          2          2
index21.index32          2          2
index22.index31          2          2
index22.index32          2          2
```

If the sum of the items to be placed in rows and columns exceeds the full dimensionality of the parameter in terms of number of sets, then a GAMS error arises. Thus in the 4 dimensional case one could not have the sum of items in rows plus columns being any greater than 4.

- The decimal control will be discussed below.
- If one is not happy with the way the indices appear then:
  - If there is dissatisfaction with the order of the sets this can only be changed by reordering the way the set indices appear by using replacement statement copying the sets into a differently defined parameter such as that below

```
parameter data2(index2,index4,index1,index3);
      data2(index2,index4,index1,index3)
      = data(index1,index2,index3,index4);
```

  - If dissatisfied with the order the set elements appear or their capitalization then one needs to understand the rules for these items and manipulate them or trick GAMS as discussed in the Rules for Item Capitalization and Ordering chapter.

7.4.2.2.2 Taking control of display decimals

A potential frustration with GAMS display output involves numerical formatting. Consider the example **dispnum.gms**. There I define the table DATA with rather disparate numbers. A resultant display of DATA yields

```
----      8 PARAMETER DATA
          index21     index22
index11 1.000000E-5 1.000000E+7
index12       3.720     200.100
```

In that display GAMS mixes together numbers in exponential and normal format with the default being a print out of three decimal places. If such a display is unsatisfactory, there are several ways of altering its appearance.

- One can alter the number of decimals using the item specific option (**dispnum.gms**)

```
option data:1:1:1;
```

which displays this item with one decimal place, yielding

```
----     10 PARAMETER DATA
```

© 2022 Prof. Bruce McCarl

```
                index21     index22
index11 1.000000E-5  10000000.0
index12         3.7        200.1
```

Here, GAMS overrides the decimals choice to insure the small number does not become zero, but I don't get the exponential display for the large number as it fits.

- One can alter the number of decimals using a global option statement to change the default decimals in all subsequent displays not subject to the specific option command just discussed. For example, the following alters the default display and all subsequent displays to 2 decimal places but would not alter the display of the item called data as it already was subject to an item specific display formatting option command. (**dispnum.gms**)

```
option decimals=2;

----     13 PARAMETER data2

            index21     index22

index11 1.000000E-5 10000000.00
index12        3.72      200.10
```

Note, GAMS still overrides the decimals choice to insure the small number does not become zero.

- One can suppress small numbers in the display manually. For example, using (**dispnum.gms**)

```
data2(index1,index2)$(data2(index1,index2) lt 0.01)=0;
```

sets all numbers to zero which are less than 0.01, yielding

```
----     14 PARAMETER DATA2
                   index21     index22
index11       _____    10000000.0
index12         3.7        200.1
```

Note one needs to employ absolute value if negative numbers are present i.e. using a command like:

```
data2(index1,index2)
$(abs(data2(index1,index2) lt 0.01))=0;
```

- Users may wish to cap the value of large numbers. I can cause the output to have the entry infinity for anything greater than the number 10,000 using (**dispnum.gms**)

```
data2(index1,index2)
      $(data(index1,index2) gt  10000)=inf;
```

© 2022 Prof. Bruce McCarl

yielding

```
----     18 PARAMETER DATA2
            index21     index22
index11 1.000000E-5        +INF
index12       3.720     200.100
```

- Users may wish to round numbers using syntax like (**dispnum.gms**)

```
data2(index1,index2)=round(data(index1,index2),0);
```

- Users may desire a report of percentage changes. These first need to be calculated in a manner such as

```
data4(index1,index2)$data2(index1,index2)=
  100*(data3(index1,index2)/data2(index1,index2)-1);
data4(index1,index2)
        $(abs(data4(index1,index2)) lt 0.1)=0;
data4(index1,index2)$(data2(index1,index2) eq 0)= na;
```

Here I set small percentage changes to zero but are careful to use absolute values so negative changes are not zeroed out. I also report numbers that would report percentage changes from a base of zero with the coding "na". The result is (**dispnum.gms**)

```
----     29 PARAMETER DATA4
            index21     index22
index11          NA
index12        80.6         1.5
```

7.4.2.2.3 Controlling item ordering

GAMS often orders things in a fashion that one dislikes. The basic rule is first in first out, so if you wish the order to be different reorder the first appearance of the item or set element. The Rules for Item Capitalization and Ordering chapter discusses the practices followed and how to change things.

7.4.2.2.4 Controlling item capitalization

GAMS users are sometimes frustrated with the capitalization of set elements or item names. The basic rule is first appearance in terms of capitalization is the scheme that will be used. Thus you wish the capitalization to be different fix up the first appearance of the item or set element so it is as desired. The chapter on Rules for Item Capitalization and Ordering discuss the rules GAMS follows.

7.4.3 Formatting pages and lines

A number of commands discussed in the Dollar Commands chapter can be used to format pages or lines. These are listed below with a brief explanation

$Double	Starts double spacing listing of subsequent echo print lines in LST file
$Eject	Start a new page in LST file
$Lines	Start new page if less than n lines are left on a page

© 2022 Prof. Bruce McCarl

$Single	Start single space listing of subsequent echo print lines in LST file
$Stitle	Define subtitle for LST file
$Title	Define LST file title

7.4.4 Output via put commands

Users can find that GAMS displays are inadequate for output presentation. A more customized output can be created using GAMS put commands. However, with this control comes a cost. Put commands involve an increased degree of technical programming. Put commands are fully discussed in the chapter Output via Put Commands.

7.4.5 Reordering set order in output

Sometimes one needs to reorder the data. Here I do it using an order set and put files in the example **reorder.gms**

```
Set I /1*4/;
set j /a1*a3
       a4 this is element 4
       a5 has a crummy name/;
set newnames(j) /a1 nuts
                 a2 bolts
                 a3 cars
                 a4 trucks
                 a5/;
table data(i,j) data to be put
    a1   a2   a3  a4   a5
1   11   12
2             14  15
3   1                   1
4         2       4.10   ;
put // 'Data as originally ordered' / @29
loop(j,put newnames.te(j):12 ' '); put /;
loop(i, put i.te(i):20 ' ';
   loop(j,if((not sameas(j,'a4')),put data(i,j):12:0 ' ';);
        if(sameas(j,'a4'),put data(i,j):12:4 ' ';)); put /);
set iwantord /o1*o100/;
set ordit(iwantord,j) / o1.a4,o2.(a1,a3),o3.a4/;
put // 'Data as reordered' / @29
loop(ordit(iwantord,j),put newnames.te(j):12 ' ');put /;
loop(i,           put i.te(i):20 ' '
loop(ordit(iwantord,j),
if((not sameas(j,'a4')),put data(i,j):12:0 ' ';);
if(sameas(j,'a4'),put data(i,j):12:4 ' ';));put /);
```

Data as originally ordered

```
                Nuts          Bolts          Cars           Trains        a5
1                11            12             0             0.0000        0
```

© 2022 Prof. Bruce McCarl

```
2              0              0             14             15.0000        0
3              1              0              0              0.0000        1
this one is 4  0              2              0              4.1000        0
```

Data as reordered (note I dropped a5 on purpose)

```
                    Trains         Nuts       Cars           Trains
1                   0.0000          11           0            0.0000
2                  15.0000           0          14           15.0000
3                   0.0000           1           0            0.0000
this one is 4       4.1000           0           0            4.1000
```

7.4.6 Preprogrammed table making utility: Gams2tbl

Rutherford has developed a preprogrammed table making utility using put commands called Gams2tbl.gms that he documents and distributes through *http://www.mpsge.org/inclib/gams2tbl.htm* Gams2tbl is a gms file that users can include in their program through a Libinclude or Batinclude that contains numerous formatting options. Gams2tbl and some of its formatting capabilities are illustrated in the following example (**canput.gms**)

```
set columns / a Horses,b Cows,c Chickens/
set rows /r1 Housing,r2 Land,r3 Feed/
table data (rows,columns) Table with default formatting
              a            b              c
r1            1          14.8233        12.99
r2            2           12              2.2
r3            3           11              3.2;
file ruthput ; put ruthput ;
$libinclude GAMS2tbl
$libinclude GAMS2tbl data
parameter roworder(rows) /r1 3,r2 1,r3 2/;
parameter colorder(columns) /a 2,b 3, c 1/  ;
$setglobal row_order roworder
$setglobal col_order colorder
$setglobal title "Table 2 where item ordering is controlled"
$libinclude GAMS2tbl data
$setglobal row_label rows
$setglobal col_label columns
$setglobal title "Table 3 where labels for sets are used"
$libinclude GAMS2tbl data
parameter decimals(columns) /a 0,b 4 ,c 1/  ;
$setglobal c_decimals decimals
$setglobal title "Table 4 where decimals are controlled"
$libinclude GAMS2tbl data
```

Yields the resultant output

© 2022 Prof. Bruce McCarl

```
Table 1 with default formatting

            a           b
r1       1.00       14.82
r2       2.00       12.00
r3       3.00       11.00

Table 2 where item ordering cor

            c           a
r2       2.20        2.00
r3       3.20        3.00
r1      12.99        1.00
```

```
Table 3 where labels for set ite

            Chickens     Horses
Land            2.20       2.00
Feed            3.20       3.00
Housing        12.99       1.00

Table 4 where decimals are contr

            Chickens     Horses
Land             2.2          2    12
Feed             3.2          3    11
Housing         13.0          1    14
```

Tables can be made in regular text, HTML or Latex formats.

7.4.7 Output to other programs

There are cases where one wishes to save things to other programs. This is generally done using Put files as discussed in the Output via Put Commands and the Links to Other Programs Including Spreadsheets chapter.

7.4.8 Obtaining graphical output

Statements may be entered into a GAMS program that permit graphical displays of data computed during a GAMS run directly in a window on a PC using Rutherford's Gnuplot or Uwe Schneider's gnuplotxyz. To graph data in a GAMS program I need to do three basic things.

- Download Schneider's *gnuplotxyz* or Rutherford's *Gnuplot* software getting both the gms and windows Gnuplot executable.
- Fill an internal array. In the gnuplotxyz example **simplegr.gms** I fill graphdata describing two lines where first dimension is name of line, second number of point on line, and third the x and y data. Such statements appear below.

```
LINES      Lines in graph /A,B/
POINTS     Points on line /1*10/
ORDINATES  ORDINATES      /X-AXIS,Y-AXIS/  ;
TABLE GRAPHDATA(LINES,POINTS,ORDINATES)
       X-AXIS   Y-AXIS
A.1         1        1
A.2         2        4
A.3         3        9
A.4         5       25
A.5        10      100
B.1         1        2
B.2         3        6
B.3         7       15
```

© 2022 Prof. Bruce McCarl

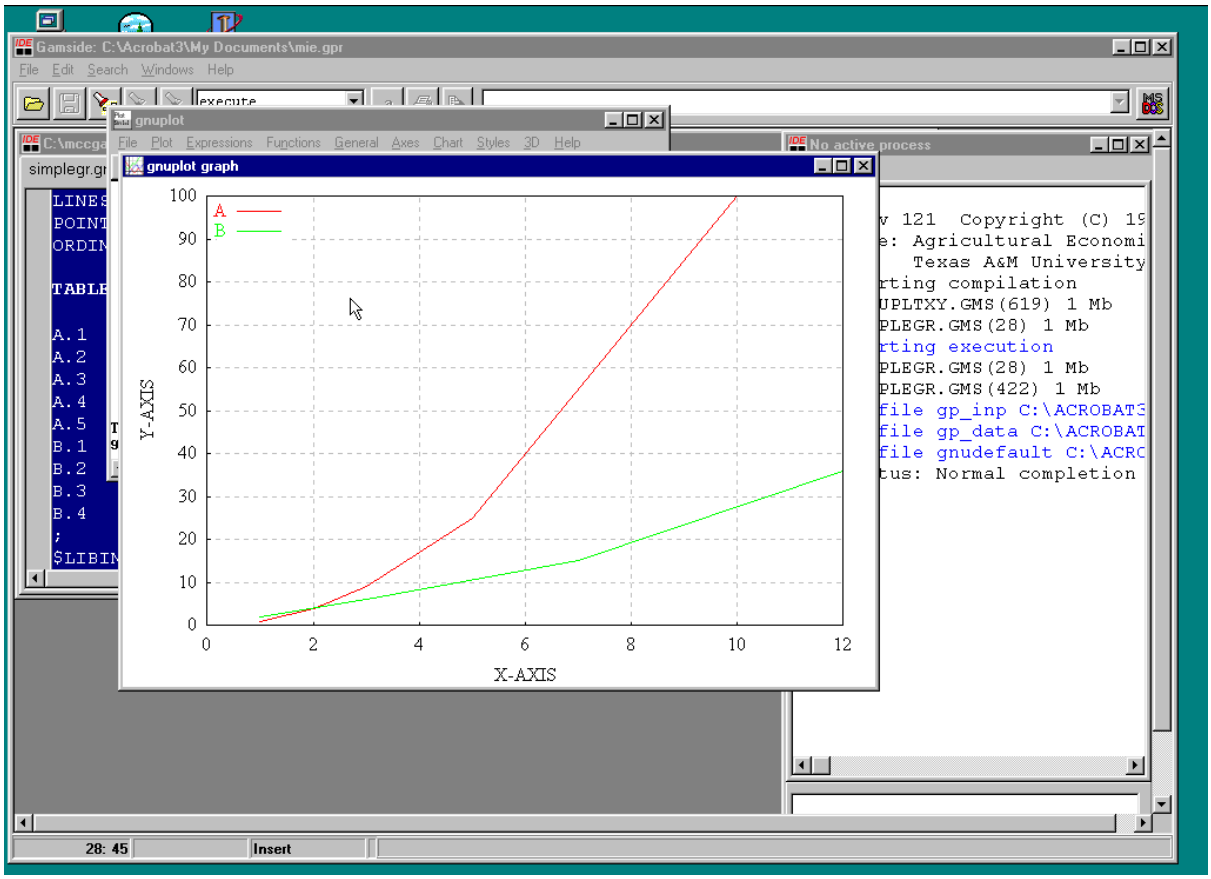
```
B.4       12       36;
```

- Then given the data call gnuplotxyz which I achieve through the GAMS statement

```
$LIBINCLUDE gnuplotxyz GRAPHDATA Y-AXIS X-AXIS
```

  where the first argument after gnuplotxyz gives the array name, the second name of a set element in the third array position which contains the data coordinates for the y axis and the third the name of a set element in the third array position which contains the data coordinates for the x axis.

In turn when I run I get two new windows that automatically open in front of the IDE



The window labeled Gnuplot graph is the graph of the data and the window labeled Gnuplot results from the execution of the source Gnuplot program. Use of such graphing is more extensively discussed and illustrated in the Links to Other Programs Including Spreadsheets chapter.

7.4.9 Sorting output

Suppose you are solving a model which is generates some output you would like to have in sorted order (arrayed from high to low). This can be done either directly in GAMS or by using Rutherford and van der Eijk's RANK libinclude procedure.

© 2022 Prof. Bruce McCarl

Sorting in GAMS
Rank

7.4.9.1 Sorting in GAMS

GAMS code maybe employed to sort output as follows. (**sorted.gms**)

```
set i /a1*a6/;
alias(i,j);
parameter unsort(i)/a1 22,a2 33,a3 12,a4 15,a5 47,a6 22/;
set asord /1*1000/;
parameter gg(i);
gg(i)=sum(j$(unsort(j)>unsort(i)),1);
set orddat(asord,i);
orddat(asord,i)$(ord(asord)=(gg(i)+1))=yes;
file sorted;
put sorted;
loop(asord,
 if(sum(orddat(asord,i),1)=1,
  put 'In place ' asord.tl:0 ' with a value of '
   loop(orddat(asord,i),
     put unsort(i):0:0 ' is item ' @42 i.tl:0/))
     if(sum(orddat(asord,i),1)>1,
         put 'In place ' asord.tl:0 ' with a value of '
           smax(orddat(asord,i),unsort(i)):0:0 ' are items '
           loop(orddat(asord,i),put @42 i.tl:0 ' ' /);)     );
```

The result

```
In place 1 with a value of 47 is item    a5
In place 2 with a value of 33 is item    a2
In place 3 with a value of 22 are items  a1
                                         a6
In place 5 with a value of 15 is item    a4
In place 6 with a value of 12 is item    a3
```

This code counts the number of entries that have a larger value than the current one and then orders items in decreasing order using another dimension that gives their relative position.

7.4.9.2 Rank

One may also use *Rutherford and van der Eijk's RANK libinclude procedure* to obtain an array giving the relative sorted position of elements. This procedure implements an *O(nlog(n))* algorithm for ranking one-dimensional numeric data within a *GAMS* program. The routine uses the GDX facility and an external program to sort the data.

The syntax for using rank is

© 2022 Prof. Bruce McCarl

```
$LIBINCLUDE rank arraytosort setofelements rankofitem optionalpercentile
```

The first three arguments are required. The last is optional. These are defined as following:

arraytosort	Name of one dimensional parameter of values to be ranked which defined over the set setofelements
setofelements	Name of one-dimensional set which is the domain of array arraytosort.
Rankofitem	Name of one dimensional parameter that after execution will contain integers giving the rank order of each element ranking from smallest to largest.
optionalpercentile	Name of one dimensional parameter that after execution will contain linearly interpolated percentiles.

Example:

GAMS code implementing RANK appears at the bottom of **sorted.gms** and is as follows

```
$LIBINCLUDE rank unsort i rankdata
put // 'After rank which sorts low' //;
loop(asord,
 loop(i$(rankdata(i)=ord(asord)),
  put 'In place ' asord.tl:0 ' with value of ' unsort(i):0:0 ' is item '
            @42 i.tl:0/));
display r;
```

The resultant output is

```
After rank which sorts low

In place 1 with a value of 12 is item    a3
In place 2 with a value of 15 is item    a4
In place 3 with a value of 22 is item    a6
In place 4 with a value of 22 is item    a1
In place 5 with a value of 33 is item    a2
In place 6 with a value of 47 is item    a5
```

Notes:

Rutherford and van der Eijk state:

- RANK only works for numeric data. You cannot sort sets.
- The first invocation must be outside of a loop or if block. This routine may be used within a loop or if block only if it is first initialized with blank invocations.
- The names rank_tmp, rank_u, and rank_p are used within these routines and may not be used in the calling program:
- This routine returns *rank* values and does not return *sorted* vectors, however rank values may be used to produce a sorted array. This can be done using computed "leads" and "lags" in GAMS' ordered set syntax, as illustrated in their *examples*.

© 2022 Prof. Bruce McCarl

7.5 Rules for Item Capitalization and Ordering

GAMS follows fixed procedures with respect to output formatting and ordering which can be expressed as rules. Knowledge of these rules allows the user to better control the capitalization and ordering attributes of the output. The topics I cover here involve

Item capitalization
Set element order
Reviewing set element ordering: $Onuellist

7.5.1 Item capitalization

When using GAMS, the capitalization format used it will employ for an item in the output **is identical to the first capitalization structure seen** for that item in the GAMS program. Thus, if a program contained "Total", "TOTAL" and "total", whichever of these appeared first would be the one used in the output. **Inside GAMS alternative capitalizations are all identical**. Available("WATER") is same as available("water") or availAble("wateR").

Examples:

Given the instructions (**dispset2.gms**)

```
Set A /Total,LINE,newone,next/;
Set b /TOTAL,LINE,NEWONE,NEXT,UNIQUETOhere/;
Scalar tT /1/;
Display b,tt;
```

The output is

```
----      11 SET       b
Total ,    LINE  ,    newone,    next  ,  UNIQUETOhere
----      4 PARAMETER tT                   =        1.000
```

where the capitalization within the set b is not as typed in defining b for any of the elements previously appearing earlier (in the set A in this case).
Similarly, the capitalization of the parameter name for tT is as it appears in the first occurrence of that name.

Notes:

- One can alter the capitalization aspects of any set element or item name in GAMS by making sure that the first entry is capitalized as desired.
- These style rules apply to all GAMS displays through display statements or put commands involving the names of set elements, sets, parameters, scalars, tables, variables, equations, models and anything else.

7.5.1.1 Reviewing capitalization: $Onsymlist and $Onuellist

Finally, one should be aware that item capitalization may be reviewed at any point by using two $ conditions. Namely typing the command

© 2022 Prof. Bruce McCarl

```
$Onsymlist
```

beginning in column 1 gives a list of all symbols in the program and they as they are capitalized. This is illustrated (with some additional formatting) for the **dispset2.gms** example below

```
SETS
  A
  b
PARAMETERS
  tT
```

Similarly, using the command in the **dispset2.gms** example

```
$Onuellist
```

beginning in column 1 gives all of the set element entries GAMS has found in the order and capitalization they will appear in the output as illustrated below.

```
Unique Elements in Entry Order
  1 Total         LINE          newone        next          UNIQUETOhere
```

7.5.2 Set element order

GAMS operates on a first in first out basis. The order of output items is always controlled by the first time something is seen. Many users are frustrated with the ordering of set elements in the output. This can be changed.

It is worthwhile knowing the rules that ordering follows. GAMS uses something called the unique element list (UEL) to store set elements. This is a single list of all set elements. The elements enter that list in the order of their appearance and that is the order in which they will appear in the output i.e. first in first out.

This means if I have the sets ONE and TWO with the elements below (**dispset.gms**)

```
set ONE /A,C,B,Total,8/;
Set TWO /D,A,F,TOTAL/;
Set items1through10 /1*10/;
Parameter  item(two) /D 1,A 3, F 5/;
item("total")=sum(two$(not sameas(two,"total")),item(two));
display item,items1through10;
```

Then the display output will appear as follows. Note in the display of the parameter **item** that the output appears with the set element **Total** in the second position even though it was typed in the last position of the set called TWO and the parameter item is defined over that set.

```
----      6 PARAMETER item
     A     3.000,    Total 9.000,    D     1.000,    F     5.000
```

© 2022 Prof. Bruce McCarl

```
----      6 SET       items1through10
8 ,   1 ,   2 ,   3 ,   4 ,   5 ,   6 ,   7 ,   9 ,   10
```

This occurs because Total was seen before D and F in the sets because it was seen first in set ONE, even though it was listed last in set TWO.

Similarly, look what has happened in the display of the set **items1through10**. Here 8 appears first then 1-7 and 9-10. This happens because the set element 8 appeared in the set ONE before 1-7 and 9-10 were seen.

Notes:

There are three ways of fixing such problems.

- One could change the name of the "total" item in the set named "two" so it was a name not seen before and 8 in the set "one" to new8 so it will not be seen again. For example I could utilize the commands (**dispset1.gms**)

```
set     ONE     /A,C,B,Total,new8/
Set     TWO     /D,A,F,TOTAL2/
Parameter item(two) /D 1,A 3, F 5/;
Item("total2")=sum(two$(not sameas(two,"total2")),item(two));
```

yielding

```
A     3.000,    D     1.000,    F     5.000 TOTAL2 9.000
```

and

```
1 ,    2 ,    3 ,    4 ,    5 ,    6 ,    7 ,    8 ,    9 ,    10
```

where use of total2 which had not been seen before and new8 which will not be seen again, restores the desired ordering.

- One could define a set at the top of the program which contains all elements **in the order desired** using set definitions as follows (**dispset3.gms**)

```
set     UELORDER    /A,B,C,D,F,Total,1*20/
set     ONE     /A,C,B,Total,8/
Set     TWO     /D,A,F,TOTAL2/
Set items1through10 /1*10/;
```

In turn the displays would always have "Total" at the end and the 1-10 in the desired order.

- One can employ an ordering set. Yet another approach can be used adding another dimension to the item of interest and employing an ordering set (**dispset4.gms**). In such a case, I add new set as the first index and put in a set that I know the order of. Then, by judiciously defining the first dimension I control the order.

Consider the following example where I want to better control appearance of the data in the table "items" on output

```
set   one         /A,B,C,D,F,Average,ITEMS/;
Set     PQ /Price,Quantity/;
Table Items(one,PQ)
       Price  Quantity
```

© 2022 Prof. Bruce McCarl

```
   A     2     9000
   B     6     3000
   C     2.5   4000
   D     2.1   3000
   F     2.4   1.90;
items("Average",PQ)= SUM(one,items(one,pq))
                     /sum(one,1$items(one,pq));
display items;
```

In turn the display output is

```
---- PARAMETER ITEMS
                Price      Quantity
     A          2.000      9000.000
     B          6.000      3000.000
     C          2.500      4000.000
     D          2.100      3000.000
     F          2.400      1.900
     Average    3.000      3800.380
```

But suppose I wanted average first. I do this by adding the set numberordr with values r1 through r100. Also I define items1 which is a new parameter with dimension one greater than items which has the numberordr set defining it in the first index position. Then I insure the items I want first have lower values in the numberordr place. (**dispset4.gms**). I also experiment with another ordering in the parameter items2.

```
Set  numberordr /r1*r100/
set  one        /A,B,C,D,F,Average/
Set  order2(numberordr,one)
                /r3.(A,D,F),r2.Average,r1.(C,B)/
Set     PQ /Price,Quantity/
Table Items(one,PQ)
      Price     Quantity
A     2         9000
B     6         3000
C     2.5       4000
D     2.1       3000
F     2.4       1.90;
items("Average",PQ)=
SUM(one,items(one,pq))/sum(one,1$items(one,pq));
parameter items1(numberordr,one,pq) ordered first way;
items1("r2",one,pq)$(not sameas(one,"average"))
=items(one,pq);
items1("r1","average",pq)=items("average",pq);
parameter items2(numberordr,one,pq) ordered second way;
items2(numberordr,one,pq)
=sum(order2(numberordr,one),items(one,pq));
display items,items1,items2;
```

© 2022 Prof. Bruce McCarl

In turn the output is

```
----PARAMETER ITEMS1                    ----     18 PARAMETER ITEMS2
                Price    Quant                                 Price    Qua
r1.Average      3.000    3800.           r1.B                  6.000    3000.
r2.A            2.000    9000.           r1.C                  2.500    4000.
r2.B            6.000    3000.           r2.Average            3.000    3800.
r2.C            2.500    4000.           r3.A                  2.000    9000.
r2.D            2.100    3000.           r3.D                  2.100    3000.
r2.F            2.400       1.           r3.F                  2.400       1.
```

where note by manipulating the association of items in the numberordr set with the things I wish reordered I can rearrange the output.

7.5.3 Reviewing set element ordering: $Onuellist

Finally, item ordering can be reviewed at any point by typing the command

```
$Onuellist
```

beginning in column 1. This causes GAMS to list all of the set element entries it has found in the order they will appear in the output as follows (**dispset.gms**)

```
Unique Elements in Entry Order

     1 A     C     B     Total   8     D
     7 F     1     2     3       4     5
     13 6     7     9     10
```

7.6 Conditionals

Frequently GAMS modelers need to be able to write expressions that operate over less than full sets or incorporate various model features conditionally depending on data. Such tasks are accomplished in GAMS using conditionals.

Basic forms of conditionals
Conditional placement and program execution speed
Forms of conditional / logical true false statements
Nesting logical conditions
The conditional alternative – the tuple

7.6.1 Basic forms of conditionals

There are four types of run time conditionals (for compile time conditionals see the Conditional Compilation chapter).

- The GAMS $ condition

© 2022 Prof. Bruce McCarl

- The If statement
- The While statement
- The Repeat statement

Each is covered below.

7.6.1.1 $ conditionals

A $ condition is placed in GAMS statements and causes an action to occur if the conditional is true. The basic $ conditional form is

```
term$logical condition
```

which says include the item **term** only if the logical condition is true. The forms of logical conditions are reviewed below. For now I will use a conditional that a named item be nonzero for illustration

```
X$(y gt 0) = 10;
```

In this case the conditional says set X=10 if the scalar y is greater than zero.

7.6.1.1.1 Ways $ conditionals are employed

$conditionals appear in a number of ways in models. Fundamentally, they are used to:

- Control whether an item is calculated-- causing calculations to occur only if certain conditions are true.
- Control the inclusion of terms in equations -- causing terms to be included only if certain conditions are true.
- Control the elements of a set that are referenced in an expression so that less than the full set of elements enters into the calculation and those that do enter only if certain conditions are true.
- Control whether equations are defined in a model -- causing selected constraint equations to only appear in the model passed to a solver only if certain conditions are true.
- Control the incidence of displays -- causing displays to occur only things only if certain conditions are true.
- Abort a program if desired.

7.6.1.1.1.1 Suppressing calculation of items (left hand side)

GAMS replacement statements involve an item to change in value on the left hand side and a set of terms that constitute the new value on the right hand side. One may cause the items to be changed only if certain conditions are met. This is controlled with conditionals. $conditionals are used to do this and may control whether the replacement calculation is done alt all or whether selected cases of it are done (only if the left hand side is defined over set elements) on a case by case basis. The general format for such a case is

© 2022 Prof. Bruce McCarl

```
namedparameter$logical condition=term;
```

or

```
namedparameter(setelementdependency)$logical condition=term;
```

and specifies that the namedparameter is set equal to the term only if the logical condition is true.

Examples:

(conditional.gms)

```
X$(qq gt 0)=3;
qq $(sum(I,q(i)) gt 0)=4;
qq $ (sum(i,abs(a(i))) gt 0)=1/sum(i,a(i));
a(i) $(qq gt 0) = q(i)+a(i);
a(i) $a(i) = q(i)/a(i);
```

All say implement this replacement statement changing the value of this parameter or its specific case only if the logical condition is true. Thus after the first statement above if X started with a value of 7 and qq was negative causing a false evaluation of the logical condition then X would finish that statement with a value of 7. On the other hand, if qq was positive then X would finish that statement with a value of 3.

Notes:

- In these cases the namedparameter value will be altered only if the logical condition is true. Otherwise, if the logical condition is false, the value of the item namedparameter will retain it's original value (or the default value of zero).
- This is known in GAMS terminology as a conditional on the **left hand side**.
- Many other logical condition forms are possible as explained below.
- The third and fifth assignments contain a common procedure where a division is done only if the denominator is non-zero.
- When the $ is on the left hand side, the equation is only computed if the logical condition is true potentially making the program faster.
- The first four cases cause the replacement statement to be entirely skipped if the conditional is false. The last conditional operates on a case specific basis since the replacement is done for the elements of i and the conditional is also dependent on i with its truth dependent upon the choice of i. Thus in the last statement a(i) is calculated only when it has a non zero value.
- The conditional in the next to last case does not include terms dependent on i so none of the a(i) will be replaced if the conditional fails.

7.6.1.1.1.2 Suppressing terms in equations (right hand side)

GAMS replacement statements involve an item to change in value on the left hand side and a set of terms that constitute the new value on the right hand side. The terms on the right hand side may be conditionally included. $conditionals are used to do this and may act on the term for all cases or on a case by case basis (only if the conditional is set element dependent). They may occur in replacement statements or in model equation specification statements (.. statements). The general format for such a case is

© 2022 Prof. Bruce McCarl

```
Namedparameter = term1+term$logical condition
```

or

```
Namedequation.. term1+term$logical condition =L= other terms;
```

and specifies that the term is added into the calculation of namedparameter or the namedequation only if the logical condition is true.

Example:

(conditional.gms)

```
qq      =  qq+1$(x gt 0);
qq      =  1$(x gt 0);
q(i)    =  a(i)+1$(a(i) gt 0);
q(i)    =  a(i)$(a(i) gt 0);
X       =  sum(I,q(i))$(qq gt 0)+4;
Eq4..      xvar+yvar$(qq gt 0)=e=3;
Eq5(i)..   ivar(i)$(a(i) gt 0)+yvar$(qq gt 0)=e=3;
```

All say include this term only if the logical condition is true.

Notes:

- In these cases the term will be included only if the logical condition is true.
- The third fourth and last cases include terms differentially depending on the set element I being referenced.
- The term may be the only one in a replacement as in the second case. In such a situation the expression is treated as if it were qq = 0 + 1$(x gt 0); where the item on the left hand side becomes zero if the condition on the right hand side fails.
- This is known in GAMS terminology as a conditional on the **right hand side**.
- Many other logical condition forms are possible as explained below.
- When the $ is placed outside a sum then the sum is only computed if the logical condition is true potentially making the program faster.
- Right hand side conditionals do not stop replacements from occurring, but left hand side ones do as elaborated on in the Calculating chapter.

7.6.1.1.1.3 Controlling indices in sums etc

It is common in algebra to want to do an operation not over all elements of a set but only over certain ones. For example if one wishes to add up ai only when ai is positive one would want a condition on the elements included in the sum as follows

$$X = \sum_{i|q_i>0} a_i$$

© 2022 Prof. Bruce McCarl

$Conditionals can be used to incorporate such restrictions. Namely they can be used to restrict the elements of a set that will be explicitly entered into a sum, loop, or other set operation (e.g. prod, smax,smin). Inclusion is controlled by a logical condition. Generally this logical condition will incorporate set element dependent terms. The general format in a SUM context is

```
sum(namedset$set dependent logical condition,  term)
```

where the above example is written as

```
X=sum(I$(q(i) gt 0),a(i));
```

where instead of sum you could use any other set operator like smin, smax or loop.

Examples:

(conditional.gms)

```
X=sum(I$(q(i) gt 0),a(i));
Loop((I,j)$(cost(I,j) gt 0),X=x+cost(I,j));
X=smin(I$(q(i) gt 0),q(i));
X=sum(I$(q(i) gt 0),1));
eq6(i)..  sum(j$(cost(i,j) gt 1),cost(i,j)*tran(i,j))=e=1;
```

All include a set element dependent case in the sum or loop or smin only if the logical condition is true. In the cases involving q(i) element I will be considered only if the data from the q parameter associated with set element I is positive.

Notes:

- The general form is that the set operator appears then the target set (or sets) is identified and then a $ appears followed by a logical condition then a term to be included in the calculation which is usually dependent on the target set(s).
- The $ only allows a term associated with an element of the set to be added or otherwise considered in the calculation if the logical condition is true.
- Many other logical condition forms are possible as explained below.

7.6.1.1.1.4 Suppressing model equations (left hand side)

Occasions occur where constraints should only be included in a model if particular conditions are met. This again can be accomplished using $conditionals. The general format for such a case is

```
equation name$logical condition..  equation specification;
```

and specifies that the named equation is defined as that given in the equation specification only if the logical condition is true.

© 2022 Prof. Bruce McCarl

Examples:

(conditional.gms)

```
Eq1$(qq gt 0)..          xvar=e=3;
Eq2$(sum(I,q(i)) gt 0).. yvar=l=4;
Eq3(i)$(a(i) gt 0)..     ivar(i)=g= -a(i);
Eq7(i)$(qq gt 0)..         sum(j,ijvar(I,j))=g= -a(i);
```

All say define this equation only if the logical condition is true.

Notes:

- In these cases, the whole named equation (see the Eq1 case above) or cases thereof when defined over a set (see the Eq3 case above) will be entered into the model only if the logical condition is true.
- The last condition on Eq7 will suppress all the cases of the equation since the conditional is not dependent on the set element i.
- This is also known in GAMS terminology as a conditional on the **left hand side**. This is essentially identical to the suppressing calculation of items in a replacement context as discussed above. Here the constraint will not be defined unless the conditional is true whereas above the execution of a replacement was determined by the conditional.
- Many other logical condition forms are possible as explained below.
- When the $ is on the left hand side the equation is only formed if the logical condition is true potentially making the program faster.

7.6.1.1.1.5 Conditionally displaying information

Conditionals may be used to cause the display of information if certain conditions are met. In such cases one usually employs a conditional in conjunction with the display command (Note the display command discussed in the Report Writing chapter). One of two forms of the command generally appear. The first involves the if syntax as discussed below where a display is executed if a particular condition is found to be true. Second, one can have the word display followed by a $condition. Therein the display will only occur if the $condition is true.

The general format for a these conditional display commands is

```
display$condition listofitems;
if(condition, display listofitems);
```

as illustrated just below (**conddisp.gms**)

```
scalar x /0/,y /10/;
display$(x+y le 0) "display when sum of x and y le 0",x,y;
x=2;
if(x gt 0,
    display "X and y here if x is positive",x,y;
```

© 2022 Prof. Bruce McCarl

```
  );
display$(x > -1) "display with display$ at second place",x;
if((x+y > 2),
    display "X and y at first place if x+y is greater than 2",x,y;
  );
if(x gt 0,display "X and y at first place if x is positive",x,y;);
```

All of these displays will only occur if the condition is true.

7.6.1.1.6 Terminating a program: Abort

Conditionals may be used to cause the execution of the program to be terminated. In such cases one employs a conditional in conjunction with the abort command. The abort command operates just like a display command with the same syntax excepting the word abort replacing display but halts the program after it's execution. The general format for a conditional abort command is

```
if(condition,abort listofitems);
```

or

```
abort$condition listofitems;
```

An example is given below (**abort.gms**)

```
scalar x /0/;
if(x > 0,abort "i stopped at first place",x;);
*note next command is redundant to above
abort$(x > 0) "i stopped with abort$ at first place",x;
display "i got past first place" ,x;
x=2;
abort$(x > 0) "i stopped with abort$ at second place",x;
*note will not get to next line
if(x > 0,abort "i stopped at second place",x;);
```

When encountered the abort command causes the job to stop with an execution error and displays the information in the command.

Abort can also be used unconditionally to stop a job just inserting the line

```
abort listofitems;
```

in the program.

7.6.1.2 If, Else, and Elseif

Another way of imposing conditionals involves use of the if statement syntax. Such statements may also involve use of else and elseif statements. In general, if statements that are not in equation specification statements can be written as $ conditions, but the use of if can make GAMS code more readable. If statements are covered in the Control Structures chapter.

© 2022 Prof. Bruce McCarl

7.6.1.3 While

Another way of imposing conditionals involves use of the while statement. In general, the while allows one to repeatedly execute a block of statements until a logical condition is satisfied. While statements are covered in the Control Structures chapter.

7.6.1.4 Repeat

Another way of imposing conditionals involves use of the repeat statement. In general, the repeat syntax causes one to execute a block of statements over and over until a logical condition is satisfied. Repeat statements are covered in the Control Structures chapter.

One can exit or jump to the end of a Repeat statement using break or continue as discussed in the context of a Loop here.

7.6.2 Conditional placement and program execution speed

As discussed above $ conditionals can be employed to cause calculations to not be done potentially causing faster executing code. For example the statement (**conditional.gms**)

```
X=sum(I,a(i)) $qq;
Eq5(i)$qq..   sum(j,ijvar(I,j))=g= -a(i);
```

will generally be faster than

```
X=sum(I$qq,a(i));
Eq6(i)..   sum(j,ijvar(I,j)) $qq =g= -a(i) $qq;
```

because some work is avoided. In general, conditionals are important tools to speed up execution. The Speeding up GAMS chapter elaborates.

7.6.3 Forms of conditional / logical true false statements

Above we used a very limited set of conditionals. A much broader set can be used. Conditionals can involve numerical relationships, the presence of numbers, sets, or acronyms. Each is slightly different and will be discussed separately.

Numerical comparisons
Data existence
Set comparisons
Acronym comparisons

7.6.3.1 Numerical comparisons

Conditionals can be formed employing logical conditions which compare two numerical expressions to see if they are equal, unequal, or if one or the other is larger. The general form is

© 2022 Prof. Bruce McCarl

```
Term$(terma operator termb)
```

The operators are as follows:

Relation	GAMS operator	Explanation
Equality	Eq or =	Does terma = termb
Not Equal	Ne or <>	Does terma ≠ termb
Greater than	GT or >	Is terma > termb
Greater or =	GE or >=	Is terma ≥ termb
Less than	LT or <	Is terma < termb
Less or =	LE or <=	Is terma ≤ termb

Each is discussed below.

7.6.3.1.1 Eq: =

One may wish to do conditional processing dependent upon whether two numerical expressions are equal or not. The form of the syntax that can be employed in such a case is

```
Terma eq termb
```

or

```
Terma = termb
```

where eq or = can be used interchangeably. Examples from **formconditional.gms**

```
If(x eq 2, z=2);
Eq1$(x=2)..  zz=e=3;
Loop(I$(sqrt(x)+1 = y+2),z=z+1)
```

7.6.3.1.2 Ne:<>

One may wish to do conditional processing dependent upon whether two numerical expressions are unequal or not. The form of the syntax that can be employed in such a case is

```
Terma ne Termb
```

or

```
Terma <> Termb
```

where ne or <> can be used interchangeably. Examples from **formconditional.gms**

```
If(x ne 2, z=2);
Eq2$(x<>2)..  zz=e=3;
Loop(I$(sqrt(x) <> y+2),z=z+1)
```

© 2022 Prof. Bruce McCarl

7.6.3.1.3 Gt: >

One may wish to do conditional processing dependent upon whether one numerical expression is greater than another or not. The form of the syntax that can be employed in such a case is

```
Terma gt Termb
```

or

```
Terma > Termb
```

where gt or > can be used interchangeably. Examples from **formconditional.gms**

```
If(x gt 2, z=2);
Eq3$(x>2)..  zz=e=3;
Loop(I$(sqrt(x) > y+2),z=z+1)
```

7.6.3.1.4 Lt: <

One may wish to do conditional processing dependent upon whether one numerical expression is less than another or not. The form of the syntax that can be employed in such a case is

```
Terma lt Termb
```

or

```
Terma < Termb
```

where Lt or < can be used interchangeably.

Examples from **formconditional.gms**

```
If(x Lt 2, z=2);
Eq4$(x<2)..  zz=e=3;
Loop(I$(sqrt(x) < y+2),z=z+1)
```

7.6.3.1.5 Ge: >=

One may wish to do conditional processing dependent upon whether one numerical expression is greater than or equal to another or not. The form of the syntax that can be employed in such a case is

```
Terma ge Termb
```

or

```
Terma >= Termb
```

© 2022 Prof. Bruce McCarl

where ge or >= can be used interchangeably.

Examples from **formconditional.gms**

```
If(x ge 2, z=2);
Eq5$(x>=2)..  zz=e=3;
Loop(I$(sqrt(x) >= y+2),z=z+1)
```

7.6.3.1.6 Le: <=

One may wish to do conditional processing dependent upon whether one numerical expression is less than or equal to another or not. The form of the syntax that can be employed in such a case is

```
Terma le Termb
```

or

```
Terma <= Termb
```

where Le or <= can be used interchangeably.

Examples from **formconditional.gms**

```
If(x Le 2, z=2);
Eq6$(x<=2)..  zz=e=3;
Loop(I$(sqrt(x) <= y+2),z=z+1)
```

7.6.3.1.7 Eqv: <=> Imp: ->

One may wish to do conditional processing dependent upon whether one numerical expression has a particular logical equivalence relationship to another or not. The form of the syntax that can be employed in such a case is

```
item1 opr item2
```

where opr is

Eqv for logical equivalence

or

<=> for logical equivalence

and

Imp for logical implication

© 2022 Prof. Bruce McCarl

or

-> for logical implication

These operate with results as follows

		Result of imp condition	Result of eqv condition
		item1 -> item2	item1 <=> item2
item1	item1	item1 imp item2	item1 eqv item2
0	0	1	1
0	1	1	0
1	0	0	0
1	1	1	1

Example:

(impeqv.gms)

```
LOOP(CASE,
  result(case,"isimp")=0;
  result(case,"isimp")=0;
  result(case,item)=data(case,item);
  IF(DATA(case,"a") imp data(case,"b")
         ,result(case,"isimp")=1;);
  IF(DATA(case,"a") eqv data(case,"b"),
          result(case,"iseqv")=1;);
);
```

7.6.3.2 Data existence

Conditionals can be formed employing logical conditions which are true if a data item exists with existence defined as the presence of a nonzero value or if a numerical expression exists (again with existence being defined as the expression result being nonzero).

7.6.3.2.1 Existence/nonzero data item or result

One may wish to do conditional processing dependent upon whether the numerical value of a scalar item, set indexed parameter or calculation result is nonzero or not. The form of the syntax that can be employed in such a case is

```
Action$Term
```

or

```
If(Term, statements);
```

or

```
While(Term, statements);
```

Examples from **dataconditional.gms**

```
Z=z+2$x;
```

© 2022 Prof. Bruce McCarl

```
If(x, z=2);
Eq5$doiwantconstraint..  zz=e=3;
Loop(I$(q(i)+q(i)**2),z=z+1)
While(x*x-1, z=z+2;x=x-1);
```

all of which will be executed if the **item being tested** in the conditional is nonzero.

This all could be expressed as **$(Term <> 0)** but this syntax is often utilized because it is more compact.

7.6.3.2.2 Computation over a set

One may wish to do conditional processing dependent upon whether the numerical value of a sum, prod, smax or smin over a set is nonzero or not. The form of the syntax that can be employed in such a case is

```
Action$Sum(set,expression)
```

or

```
If(Smin(set,expression), statements);
```

or

```
While(Smax(set,expression), statements);
```

Examples:

(dataconditional.gms)

```
Z=z+2$sum(I$q(i),1);
If(smin(I,q(i)), z=2);
Eq5$prod(I,q(i))..  zz=e=3;
Eq6(j)$sum(I,abs(data(I,j)))..  zz=e=3;
Loop(I$sum(j,abs(data(I,j))),z=z+1)
While(prod(I,q(i)), z=2;q(i)=q(i)-2);
```

Notes:

- In these cases the term subject to the conditional will be evaluated or the calculation will be executed if the final result of the set dependent expression is nonzero.
- This all could be expressed as $(Term ne 0) but the syntax $(Term) is often utilized because it is more compact.
- The conditional as in the expression sum(I$q(i),1) reveals a commonly used GAMS trick for seeing whether there is any nonzero data in an item (by counting it where the count will be zero when all the data in the vector are zero).
- Including the conditional expression $sum(j,abs(data(I,j)) is a common GAMS trick for seeing whether there is any nonzero data across all the j alternatives in a row I of data(I,j).

© 2022 Prof. Bruce McCarl

7.6.3.3 Set comparisons

Conditionals can be formed employing logical conditions which depend on set elements in terms of the relative placement of the element being worked on, the contents of the text for a set element or whether an item falls in a subset or tuple. The forms follow.

7.6.3.3.1 Element position: Ord and Card

One may wish to do conditional processing dependent upon the relative position of an element within a set. The Ord and Card functions as discussed in the sets chapter can be used in conditionals. Ord (setelement) returns the number of the element being referenced relative to the total number of elements in the set. Card(setname) gives the total number of elements in the named set. These numerical values can be subjected to any of the numerical comparisons above.

Examples:

(setconditional.gms)

If one wishes to do special things with respect to the first, last and intermediate elements in a set one could use code like

```
FIRSTBAL(period)$(ord(period) eq 1)..
        SELL(period) + STORE(period) =L= INVENTORY;
INTERBAL(PERIOD)
       $(ORD(period) GT 1 and ORD(period) lt CARD(period))..
          SELL(PERIOD) =L= STORE(PERIOD-1) - STORE(PERIOD);
LASTBAL(PERIOD)$(ORD(period) eq CARD(period))..
          SELL(PERIOD)=L= STORE(PERIOD-1);
```

or

```
loop(period
  $(ORD(period) GT 1 and ORD(period) lt CARD(period)),
      Z=z+1;);
```

Notes:

- Ord does not work with sets that contain calculated elements or are otherwise unordered, only sets with a priori explicitly specified values.
- Ord refers to the relative position of each element in the set not necessarily the order in which they are typed. In particular, the order may be different as determined by the rules for set ordering as discussed in the Rules for Item Capitalization and Ordering chapter.
- Card works with any sets whether they contain calculated elements or not.
- Testing for Ord(i) = 1 looks to see if one is on first element.
- Testing for Ord(i) = Card(i) looks to see if one is on last element.

7.6.3.3.2 Element text comparison: Sameas and Diag

One may wish to do conditional processing dependent upon the text defining a name of a set element matching the text for a particular text string or matching up with the text for a name of a set element in

© 2022 Prof. Bruce McCarl

another set. This can be done in GAMS using the sameas or diag commands. SAMEAS returns a value of true or false using the syntax

```
SAMEAS(setelement,othersetelement)
```

or

```
sameas(asetelement,"texttotest")
```

where the first sameas is true if the name of **setelement** is the same as the name of **othersetelement** and is false otherwise. Similarly, **sameas(asetelement,"texttotest")** is true if name of **asetelement** is the same as the **texttotest** and false otherwise.

DIAG works identically but returns numerical values of one if the text matches and zero otherwise.

```
Diag(setelement,othersetelement)
```

or

```
Diag(asetelement,"text")
```

Examples:

(setconditional.gms)

The following **red** uses of **sameas** and **diag** will only permit the case of I and j to be part of the sum where the elements for both the same and in this case will only work for the element named **Boston** and do not require the sets to be subsets of each other. The **blue** cases will only operate for the element of I which has the name "**new york**".

```
Set   cityI       / "new york", Chicago, boston/;
Set   cityj       /boston/;
Scalar ciz,cir,cirr;
ciZ=sum(sameas(cityI,cityj),1);
ciR=sum((cityI,cityj)$ sameas(cityI,cityj),1);
ciRR=sum(sameas(cityI,"new york"),1);
ciZ=sum((cityi,cityj)$diag(cityI,cityj) ,1);
ciRR=sum(cityi$diag(cityI,"new york"),1);
```

7.6.3.3.3 Subset or tuple membership

One may wish to do conditional processing dependent upon whether a set element is defined in a particular subset or tuple. The form of the syntax that can be employed in the subset case is

```
Action$subset(setelement)
```

or

```
If(subset("quotedsetelement"), Action);
```

or

```
While(subset("quotedsetelement"), Action);
```

or in the tuple case is

© 2022 Prof. Bruce McCarl

```
Action$tuplename(setelement, set2element)
```

or

```
If(tuplename(setelement, set2element), Action);
```

or

```
While(tuplename(setelement, set2element), Action);
```

where in both setelement is a set name which must be controlled within the action or earlier in a Loop statement and quotedsetelement is a particular set element name encased in quotes.

Examples:

(setconditional.gms)

```
ciZ=sum((allcity,cityj) $cityi(allcity),1);
ciZ$cityi("boston")=sum(allcity,1);
loop((allcity,cityj) $ tuple(allcity,cityj),
      ciz=ciz+ord(allcity)+ord(cityj)*100);
if(cityj("boston"),ciz=1);
if(tuple("orlando","boston"),ciz=1);
```

7.6.3.4 Acronym comparisons

Conditionals can be formed employing logical conditions which are based on acronyms, which are character string values, but many only involve the eq , = , ne , or <> operators. The forms are as follows:

Logical conditions. The syntax is

```
If(paramcontainingacronym op acronym ,action);
```

or

```
Action$(paramcontainingacronym op acronym)
```

where op is one of the eq , = , ne , or <> operators.

You must do comparisons with other acronyms not with text strings.

Examples:

(acronym.gms)

```
acronyms nameforit,nextone;
acronym  acronym3
acronym  doit
parameter textstrings(i)
     /i1 nameforit
      i2 nextone
      i3 acronym3/ ;
loop(i,
```

© 2022 Prof. Bruce McCarl

```
  if(textstrings(i)=nameforit,put 'Something special'););
aa(i)$(textstrings(i) =doit).. 3*x(i)=e=1;
```

7.6.4 Nesting logical conditions

Conditionals can be formed employing more than one logical condition. This is called nesting.

Nesting operators
Nested $ conditionals

7.6.4.1 Nesting operators

Sometimes it is desirable to use complex logical conditions where several things are simultaneously true or at least one of several things. This can be done by combining conditions using logical operators or complex (nested) $ conditions. The logical operators are And, Or, Xor and Not along with nested $ conditions.

And -	perform an action if two or more conditionals are true simultaneously
Or	perform an action if at least one of two or more conditionals are true
Xor	perform an action if only one of two or more conditionals are true
Not	do something when a conditional is not true
Nested $	statements can involve multiple $ conditions

Each is discussed below.

7.6.4.1.1 And

When one wishes to perform an action if two or more conditionals are true simultaneously one can join them with an **and** operator. This involves using syntax like

```
Action$(logical condition 1 and logical condition 2 and
                        Logical condition 3)
```

or

```
If((logical condition 1 and logical condition 2 and
                        Logical condition 3), Action);
```

or

```
While((logical condition 1 and logical condition 2 and
                        Logical condition 3), Action);
```

Examples:

(complexcond.gms)

```
u(k)$(s(k) and t(k)) = a(k);
u(k)$(s(k) and u(k) and t(k)) = a(k);
loop(k,if(s(k) lt 0 and t(k), u(k) = a(k)) );
```

© 2022 Prof. Bruce McCarl

Notes:

- All of the logical conditions must be simultaneously true for the total logical condition to be true.
- The **and** operator can be mixed with other **and**, **or**, **not**, **xor**, **nested $** in a complex logical condition. When this is done GAMS will execute the various statement components according to a predefined operator precedence. However it is advisable to be cautious and use parentheses to carefully control the meaning of the condition.

7.6.4.1.2 Or

When one wishes to perform an action if at least one of two or more conditionals apply one can join them with an **or** operator. This involves using syntax like

```
Action$(logicalcondition1 or logicalcondition2 or
                              logicalcondition3)
```

or

```
If(logicalcondition1 or logicalcondition2 or
                        logicalcondition3), Action);
```

or

```
While(logicalcondition1 or logicalcondition2 or
                           logicalcondition3), Action);
```

Examples:

(complexcond.gms)

```
u(k)$(s(k) or t(k)) = a(k);
u(k)$(s(k) or u(k) or t(k)) = a(k);
loop(k,if(s(k) lt 0 or t(k), u(k) = a(k)) );
```

Notes:

- Any one or more than one of the logical conditions must be simultaneously true for the total logical condition to be true.
- The **or** operator can be mixed with other **and** , **or** , **not**, **xor** , **nested $** in a complex logical condition. When this is done GAMS will execute the various statement components according to a predefined operator precedence. However it is advisable to be cautious and use parentheses to carefully control the meaning of the condition.

7.6.4.1.3 Xor

When one wishes to perform an action if and only if one of two or more conditionals apply one can join them with an **xor** operator. This involves using syntax like

```
Action$(logical condition 1 xor logical condition 2 xor
                        Logical condition 3)
```

© 2022 Prof. Bruce McCarl

or

```
If(logical condition 1 xor logical condition 2 xor
                          Logical condition 3, Action);
```

or

```
While(logical condition 1 xor logical condition 2 xor
                          Logical condition 3, Action);
```

Examples:

(complexcond.gms)

```
u(k)$(s(k) xor t(k)) = a(k);
u(k)$(s(k) xor u(k) xor t(k)) = a(k);
loop(k,if(s(k) lt 0 xor t(k), u(k) = a(k)) );
```

Notes:

- One and only one not more than one of the logical conditions can be simultaneously true if the total logical condition to be true.
- The **xor** operator can be mixed with other **and** , **or** , **not**, **xor** , **nested $** in a complex logical condition. When this is done GAMS will execute the various statement components according to a predefined operator precedence. However it is advisable to be cautious and use parentheses to carefully control the meaning of the condition.

7.6.4.1.4 Not

When one wishes to do something when a conditional is not true one can prefix it a **not** operator. This involves using syntax like

```
Action$( not logical condition 1)
```

or

```
If(not logical condition 1, Action);
```

or

```
While(not logical condition 1), Action);
```

Examples:

(complexcond.gms)

```
u(k)$(not s(k)) = a(k);
loop(k,if(not (s(k) lt 0), u(k) = a(k)) );
```

Notes:

- When a logical condition is preceded by a **not** then the logical condition must be **false** for the **not** of it to be **true**.
- The **not** operator can be mixed with other **and** , **or** , **not**, **xor** , **nested $** in a complex logical

© 2022 Prof. Bruce McCarl

condition. When this is done GAMS will execute the various statement components according to a predefined operator precedence. However it is **virtually imperative** that you set the **not** off in parentheses to insure the meaning of the statement.

7.6.4.2 Nested $ conditionals

$ conditions can be nested. The term $(logicalcondition1$(logicalcondition2)) can also be written as $(logicalcondition1 and logicalcondition2). For nested $ conditions, all succeeding expressions after the first $ must be enclosed in parentheses. Consider the following example (**complexcond.gms**),

```
u(k)$(s(k)$t(k)) = a(k) ;
```

The assignment will be made only for those members of k that are also members or associated with data in both s and t. Note the position of the parenthesis in the $ condition. The statement above can be rewritten as

```
u(k)$(s(k) and t(k)) = a(k) ;
```

To assist with the readability of statements, one should usually employ the operator and instead of nesting dollar operators.

7.6.4.2.1 Nested Operators and precedence order

Nested operators are acted on in a fixed precedence order and are also operated on in an order relative to mathematical operators. The default precedence order in the absence of parentheses is shown below in decreasing order.

Operation	Operator	Precedence
Exponentiation	**	**1**
Multiplication, Division	* /	**2**
Addition, Subtraction	+ -	**3**
Numerical Relationships	< ,<=, = , <>, >=, > lt , le ,eq, ne , ge, gt	**4**
Not	not	**5**
And	and	**6**
Or, Xor	or, xor	**7**
Logical Equivalence & Logical implication	Imp, Eqv ->, <=>	8

When operators with the same precedence appear in an expression or logical condition they are performed from left to right.

7.6.4.2.1.1 Note of caution

It is always advisable to use parentheses rather than relying on the precedence order of operators. It prevents errors and makes the intention clear. It is especially important to encapsulate not operators in parentheses to limit their scope.

© 2022 Prof. Bruce McCarl

Examples:

(complexcond.gms)

```
u(k)$(s(k) and {u(k) xor t(k)}) = a(k);
u(k)$(s(k) and {u(k) or t(k)}) = a(k);
u(k)$([not s(k)] and {u(k) or  t(k)}) = a(k);
u(k)$(s(k) xor {u(k) and t(k){) = a(k);
u(k)$(s(k) xor [not {u(k) and t(k)}]) = a(k);
```

Notes:

- You can use (),[] and {} in pairs interchangeably.
- When the **not** operator is mixed with other **and**, **or**, **not**, **xor**, **nested $** in a complex logical condition it is **virtually imperative** that you set the **not** off in parentheses to insure the meaning of the statement.
- When more than one **and**, **or**, **not**, **xor**, **nested $** operator are used in a complex logical condition GAMS will execute the various statement components according to a predefined operator precedence.

7.6.5 The conditional alternative: the tuple

Often the conditionals required in a model are complex and are used in a repetitive manner. It is sometimes simpler to establish a tuple that encapsulates the conditionals and only use that tuple instead of the complex set of conditionals.

Example:

In the model below the logical condition in red that appears repetitively can be replaced with the tuple making the model visually simpler plus potentially easier to maintain. (tuple.gms)

```
TCOSTEQ..       TCOST =E= SUM((PLANT,MARKET)
                            $(    supply(plant)
                              and demand(market)
                              and distance(plant,market))
                                  , SHIPMENTS(PLANT,MARKET)*
                                    COST(PLANT,MARKET));
 SUPPLYEQ(PLANT).. SUM(MARKET
                         $(    supply(plant)
                           and demand(market)
                           and distance(plant,market))
                                ,SHIPMENTS(PLANT, MARKET))
                                 =L= SUPPLY(PLANT);
DEMANDEQ(MARKET)..  SUM(PLANT
                          $(    supply(plant)
```

© 2022 Prof. Bruce McCarl

```
                                    and demand(market)
                                    and Distance(plant,market))
                                      ,  SHIPMENTS(PLANT, MARKET))
                                        =G= DEMAND(MARKET);
```

where the alternative with the tuple is

```
set thistuple(plant,market) thistuple expressing conditional;
thistuple(plant,market)$(    supply(plant)
                         and demand(market)
                         and distance(plant,market))
             =yes;
TCOSTEQ2..          TCOST =E= SUM(thistuple(plant,market)
                                  , SHIPMENTS(PLANT,MARKET)*
                                    COST(PLANT,MARKET));
SUPPLYEQ2(PLANT).. SUM(thistuple(plant,market)
                                 ,SHIPMENTS(PLANT, MARKET))
                                 =L= SUPPLY(PLANT);
DEMANDEQ2(MARKET)..  SUM(thistuple(plant,market)
                                 ,  SHIPMENTS(PLANT, MARKET))
                                   =G= DEMAND(MARKET);
```

Notes:

- More on tuples appears in the Sets and Calculating chapter.
- One should be careful here with the fact that a tuple is only calculated in a static manner and the calculation will only be updated if it is reissued as discussed in the Calculating chapter.

7.7 Control Structures

There is a class of statements in GAMS that control the number of times a group of statements are executed. The If syntax controls whether a single use occurs. The For, While, Loop and Repeat statements permit multiple executions of a set of statements. Each is discussed here.

If, Else, and Elseif
Loop
While
For, To, Downto, and By
Repeat, Until

7.7.1 If, Else, and Elseif

Another way of imposing conditionals involves use of the If statement which also involves the else and Elseif statements. The optional else part allows specification for cases where the If fails. The Elseif part allows an alternative If test to be presented relative to the original.

- The basic syntax for an If statement without an else or Elseif is:

```
If (logical condition,
```

© 2022 Prof. Bruce McCarl

```
        statements to be executed If true ;);
```

- The basic syntax for an If statement with an else is:

```
If (logical condition,
      statements executed If condition true;
else
      statements execut. If cond. not true;);
```

- The basic syntax for an If statement with an Elseif is:

```
If (logical condition,
      statements to be executed If true ;
Elseif logical condition,
      statements executed If this conditional
      is true and the earlier one is false );
```

- The basic syntax for an If statement with two Elseif s is:

```
If (logical condition,
      statements to be executed If true ;
Elseif logical condition,
      statements executed If conditional is
      true and the earlier one is false ;
Elseif logical condition,
      statements executed If conditional is
      true and both the earlier ones are false
);
```

An else could also be added to 3 or 4 that would be executed when all the previous conditionals were not satisfied.

Examples:

(ifelseitelse.gms)

A simple If

```
If (key <= 0,
     data1(i) = -1 ;
     key2=case1;
) ;
```

A simple if when **$onend** is present

```
$onend
If key <= 0 then
     data1(i) = -1 ;
     key2=case1;
endif ;
```

An If with an else

```
If (key <= 0,
     data1(i) = -1 ;
  key2=case1;
else
  data1(i) = data1(i)**3 ;
```

© 2022 Prof. Bruce McCarl

```
    key2=case4;
) ;
```

A complex If with more than one Elseif and an else

```
If (key <= 0,
     data1(i) = -1 ;
     key2=case1;
Elseif ((key > -1) and (key < 1)),
     data1(i) = data1(i)**2 ;
     key2=case2;
Elseif ((key >= 1) and (key < 2)),
     data1(i) = data1(i)/2 ;
     key2=case3;
else
     data1(i) = data1(i)**3 ;
     key2=case4;
) ;
```

An If else statement group that contains solve statements.

```
solve ml using lp maximizing z;
If ((ml.modelstat eq 4),
      display 'model ml was infeasible',
              'relaxing bounds on x and solving again';
   x.up(i) = 2*x.up(i) ;
   solve ml using lp minimizing z ;
else
   If ((ml.modelstat ne 1),
      abort "error solving model ml" ;
      );
);
```

Notes:

- When $onend is not present the **If** is followed by an open (and a close) which surround the logical condition and the subsequent statements to be executed. The logical condition is followed by a comma.
- When $onend is present the **If** is followed by a logical condition a **then** followed by the subsequent statements to be executed. The statement is ended with an **endif**.
- The statements executed by an If are ended by the appearance of an **Elseif** or an **else**.
- One cannot place parameter, acronym, set, file, table, model, equation, variable or scalar statements or .. equation declarations in the statements to be executed when the If logical condition is true.
- Many other logical condition forms are possible as explained in the Conditional chapter.
- If statements usually can also be written as a set of dollar conditionals, but the If statement may make the code more readable.
- The body of the If statement can contain solve statements.

© 2022 Prof. Bruce McCarl

7.7.1.1 Alternative syntax

A dollar command can be used to alter the syntax for the If statement. Namely, Endif is introduced as a keyword when $Onend is active and becomes illegal upon use of $Offend. When active, Endif ends the If statement. The dollar command option is employed using the syntax

```
$Offend
```

or

```
$Onend
```

Setting the $Onend dollar command will make the alternative syntax valid, but makes the standard syntax invalid.

7.7.1.1.1 Endif

Ordinarily Ifs are of the form

```
If(conditional,
  statements ;
) ;
```

and when $Onend is specified the statement becomes

```
$Onend
If conditional then
     statements ;
Endif;
```

Example:

(control.gms)

The following two commands are equivalent and illustrate the ways the syntax varies.

```
If (x ne 0,
     DATA(I)=12 ;
  );
$Onend
If x ne 0 then
     DATA(I)=12 ;
Endif;
```

7.7.2 Loop

The Loop statement allows one to execute a group of statements for each element of a set. The syntax of the Loop statement is,

```
Loop((sets_to_vary),
      statement or statements to execute
);
```

© 2022 Prof. Bruce McCarl

If the sets_to_vary contains one set, then the statement can be

```
Loop(set_to_vary,
        statement or statements to execute
);
```

The Loop statement causes GAMS to execute the **statement or statements** for each member of sets_to_vary in turn. The order of evaluation is determined by the contents of the UEL list as discussed in the Rules for Item Capitalization and Ordering chapter. One can also break out of loops using Break and Continue as discussed below.

Example:

The following syntax would cause the model to be solved for each element within the set I. In that Loop I first revise some data in the model in accordance with the ith element of savparam. In turn the objective value would be saved in the ith element of data.

```
Loop (i,
  problemdata=savparam(i);
  Solve mymodel using lp maximizing profit;
  Data(i)=profit.l;
) ;
```

Notes:

- Within the Loop the index is treated as If it referenced only the single set element that the Loop index takes on during each pass. Thus If the set I in the example above had the elements corn and wheat the first pass would act as If it were

  ```
  problemdata=savparam("corn");
  Solve mymodel using lp maximizing profit;
  Data("corn")=profit.l;
  ```

  and on the second pass as If it were

  ```
  problemdata=savparam("wheat");
  Solve mymodel using lp maximizing profit;
  Data("wheat")=profit.l;
  ```

- The Loop set elements addressed may be subject to a conditional as discussed in the Conditionals chapter.
- Loops may involve more than one set e.g. Loop((I,j), statement or statements to execute);
- One cannot include parameter, acronym, set, file, table, model, equation, variable or scalar statements or .. equation declarations inside a Loop statement.
- It is illegal to modify any set indexed by the Loop statement inside the body of the Loop.
- Solves can appear within a Loop and Loops are often conveniently used to do a scenario based study with repeated solves.

© 2022 Prof. Bruce McCarl

- A Loop is often used to perform iterative calculations.
- Break causes the Loop to be terminated and can have an argument where break **1** causes the innermost Loop to terminate, break 2 the next most inner loop and so forth. The example Breakcontinue.gms illustrates use of this. This also works within WHILE, REPEAT and FOR commands.
- Continue causes execution to jump to the end of a Loop. The example Breakcontinue.gms illustrates use of this. This also works for WHILE, REPEAT and FOR commands.

7.7.2.1 Alternative syntax

A dollar command can be used to alter the syntax for the Loop statement. Namely, Endloop is introduced as a keyword when $Onend is active and becomes illegal upon use of $Offend. When active, Endloop ends the Loop statement. The dollar command option is employed using the syntax

```
$Offend
```

or

```
$Onend
```

Setting the $Onend dollar command will make the alternative syntax valid, but makes the standard syntax invalid.

7.7.2.1.1 Endloop

Ordinarily Loop statements require syntax of the form

```
Loop (settovary,
  statements ;
) ;
```

and when $Onend is specified the required syntax becomes

```
$Onend
Loop settovary do
     statements ;
Endloop;
```

Example:

(control.gms)

The following two commands are equivalent and illustrate the ways the syntax varies.

```
Loop (i,
  DATA(I)=12 ;
) ;
$Onend
```

© 2022 Prof. Bruce McCarl

```
Loop i do
     DATA(I)=12 ;
Endloop;
```

7.7.3 While

The While statement allows one to repeatedly execute a block of statements until a logical condition is satisfied. Ordinarily, the syntax of the While statement is:

```
While (logical condition,
     statements to be executed While condition is true;
     );
```

But when $Onend is specified the statement becomes

```
$Onend
While conditional do
        statements ;
Endwhile;
```

One can exit or jump to the end of a While statement using break or continue as discussed in the context of a Loop here.

Examples:

A binary root finder using While (**while.gms**) is as follows

```
While (converge = 0 and iter lt lim,
      root=(maxroot+minroot)/2;
      iter=iter+1;
      function_value=a-b*root+c*sqr(root);
      If(abs(function_value) lt tolerance,
            converge=1;
      else
            If(sign(function_value1)=sign(function_value),
                  minroot=root;
                  function_value1=function_value;
            else
                  maxroot=root;
                  function_value2=function_value;
            );
       );
);

or

$onend
While converge = 0 and iter lt lim do
```

© 2022 Prof. Bruce McCarl

```
        root=(maxroot+minroot)/2;
        iter=iter+1;
        function_value=a-b*root+c*sqr(root);
        function_value=function_value;
        If(abs(function_value)) lt tolerance then
            converge=1;
        else
           if(sign(function_value1)=sign(function_value)) then
              minroot=root;
              function_value1=function_value;
           else
              maxroot=root;
              function_value2=function_value;
           endif;
        endif;
    endwhile;
```

Notes:

- When $onend is not present While is followed by an open (and a close) which surround the logical condition and the subsequent statements to be executed. Furthermore the logical condition is followed by a comma.
- When $onend is present While is followed by a logical condition possibly in parentheses then a do and the subsequent statements to be executed. Finally the statement is ended with an Endwhile.
- One cannot place parameter, acronym, set, file, table, model, equation, variable or scalar statements or .. equation declarations in the in the statements to be executed If condition is true.
- Many other logical condition forms are possible as explained below.
- The total number of passes through the While statements can be limited using the option Forlim.

7.7.3.1 Alternative syntax

A dollar command can be used to alter the syntax for the While statement. Namely, Endwhile is introduced as a keyword when $Onend is active and becomes illegal upon use of $Offend. When active, Endwhile ends the While statement. The dollar command option is employed using the syntax

```
$Offend
```

or

```
$Onend
```

Entering the $Onend command will make the alternative syntax valid, but makes the standard syntax invalid.

7.7.3.1.1 Endwhile

Ordinarily, While statements are of the form

© 2022 Prof. Bruce McCarl

```
While(conditional,
  statements ;
) ;
```

and when $Onend is specified the statement becomes

```
$Onend
While conditional then
     statements ;
Endif;
```

Example:

(control.gms)

The following two commands are equivalent and illustrate the ways the syntax varies.

```
While(x<10,
    x=x+0.01;
     );
$Onend
While x<10 do
     x=x+0.01;
    Endwhile;
```

7.7.4 For, To, Downto, and By

The For statement allows one to repeatedly execute a block of statements over a successively varied values of a scalar.

```
for (scalarq = startval to(downto) endval by increment,
     statements;
     );
```

where scalarq is a scalar
startval is the constant or scalar giving the value where scalarq will begin
endval is a constant or scalar giving the value that will result in statement termination when scalarq equals or passes it
increment is a positive constant or scalar which is optional and defaults to one
To indicates that GAMS will add increment until scalarq gets equal to or larger than end
Downto indicates that GAMS will subtract increment until scalarq gets equal to or smaller than end.

One can exit or jump to the end of a For statement using break or continue as discussed in the context of a Loop here.

Example:

(for.gms)

© 2022 Prof. Bruce McCarl

```
for (iter = 1 to iterlim,
  root=(maxroot+minroot)/2;
  function_value=a-b*root+c*sqr(root);
  If(abs(function_value) lt tolerance,
      iter=iterlim;
  else
      If(sign(function_value1)=
         sign(function_value),
         minroot=root;
         function_value1=function_value;
      else
         maxroot=root;
         function_value2=function_value;);
      );
  );
```

Notes:

- One cannot incorporate parameter, set, file, table, model, equation, variable or scalar statements or .. equation declarations inside a for statement.
- The values of start, end and increment need not be integer. The start and end values can be positive or negative real numbers.
- The value of increment has to be a positive real number.
- The total number of passes through the For statements can be limited using the option Forlim.

```
option forlim=10;
```

7.7.4.1 Alternative syntax

A dollar command can be used to alter the for statement syntax. Namely, Endfor is introduced as a keyword when $Onend is active and becomes illegal upon use of $Offend. When active, Endfor ends the For statement. The dollar command option is employed using the syntax

```
$Offend
```

or

```
$Onend
```

Setting the $Onend dollar command option will make the alternative syntax valid, but makes the standard syntax invalid.

7.7.4.1.1 Endfor

Ordinarily For statements are of the form

```
For(scalar=limits,
  statements ;
) ;
```

© 2022 Prof. Bruce McCarl

and when $Onend is specified the statement becomes

```
$Onend
For scalar=limits do
     statements ;
endfor;
```

Example:

(control.gms)

The following two commands are equivalent and illustrate the ways the syntax varies.

```
for(x=1 downto 12 by 2,
    data(i)=x;
   );
$Onend
for x=1 downto 12 by 2do
    data(i)=x;
   endfor;
```

7.7.5 Repeat, Until

The Repeat statement causes one to execute a block of statements over and over until a logical condition is satisfied. The syntax of the Repeat statement is:

```
repeat ( statements to be executed;
   until logical condition is true );
```

Examples:

A binary root finder using Repeat may be found in **repeat.gms**

```
repeat (
  root=root+inc;
  function_value2= a-b*root+c*sqr(root);
  If((sign(function_value1) ne sign(function_value2)
    and abs(function_value1) gt 0
    and abs(function_value2) gt tolerance),
      maxroot=root;
      signswitch=1
  else
    If(abs(function_value2) gt tolerance,
      function_value1=function_value2;
      minroot=root;));
  until (signswitch>0 or root > maxroot) ;);
```

Notes:

© 2022 Prof. Bruce McCarl

- Repeat is followed by an open (and a close) which surround subsequent statements to be executed, an until and the logical condition which when true causes statement termination.
- The until and logical condition are at the end of the statement.
- The until precedes the logical condition and is then followed by a closing parenthesis.
- One cannot place GAMS set, acronym, for, scalar, parameter, table, variable, equation or model statements inside the repeat statement.
- Many logical condition forms are possible as explained in the Conditionals chapter.
- The total number of passes through the Repeat statements can be limited using the option Forlim.

8 Doing a Comparative Analysis with GAMS

Most of the modeling done today in my professional field involves comparative analysis. Models, once built, are almost always subjected to a number of alternative scenarios where the analyst compares the results across solutions to see what the effect is of the various scenario assumptions. This chapter covers how to do comparative analysis with GAMS.

Basic approaches
Manual approach
An automated approach - avoiding repeated work
Where am I?

8.1 Basic approaches

There are two ways of accomplishing scenario analysis. One can

- Use multiple GAMS submissions or multiple solves generating report writing output then manually comparing the analysis results.
- Use the GAMS loop procedure and set up a comparative scenario analysis system that creates cross scenario comparison tables.

I will only briefly touch on the first approach but will extensively cover the second approach.

8.2 Manual approach

Suppose we have an existing model and we wish to do a comparative analysis altering commodity prices. In this case we will use as a base a farm profit-maximizing model that is called **farmcomp.gms**. Within that model we have a vector of commodity prices

```
PARAMETER price(primary)  prices for products
          /corn     2.20
          soybeans 5.00
          beef     0.50/
```

and we wish to run a base case and two cases with alternative prices – a case with the beef price at

© 2022 Prof. Bruce McCarl

$0.70 and one with the corn price at $2.70. Suppose we also have programmed a report writer using the techniques discussed in the Improving Output via Report Writing chapter and placed the calculations for it in the file **farmrep.gms**. When that file is included using the $Include syntax as discussed in Including External Files chapter we get a report on whatever solution was generated by the last solve executed in the GAMS program. This report consists of a number of tables. We will only focus on the one below

```
Set alli   allitems
           /Corn,Soybeans,Beef  ,cattle
            Water,Cropland,Pastureland
            Fertilizer,Seed,Othercost, Veternary, Supplement
           "April Labor","May Labor"
           "Summer Labor","Sept Labor","Oct Labor",
            Cattlefeed
            Total/
Set measures  output measures
     / "Net Income", "Land use", "Dry Cropping", "Irr Cropping",
       "Livestock", "Resource Value","Product Value"/
Parameter summary(alli,measures)   Farm Summary;
```

which computes the report summary which contains rows for the major commodities as named in the alli set and the columns identified in the measures set.

Now given this code we can accomplish a comparative analysis in the file **mancomp.gms** as follows

```
$include Farmcomp.gms
display price;
$include Farmrep.gms
price("beef")=0.70;
SOLVE Firm USING LP MAXIMIZING NETINCOME;
display price;
$include Farmrep.gms
price("corn")=2.70;
SOLVE Firm USING LP MAXIMIZING NETINCOME;
display price;
$include Farmrep.gms
```

This code

- sets up and solves the original model using the file **farmcomp.gms** which also contains the set definitions for the report
- displays the initial prices
- constructs a report by including **farmrep.gms**
- alters the beef price to $0.70
- solves the altered model
- constructs a report on the altered model by including **farmrep.gms**

© 2022 Prof. Bruce McCarl

- alters the corn price to $2.70
- solves the altered model
- constructs a report on the altered model by including **farmrep.gms**

This yields in the LST file a display arising from line 287 which is associated with the Base model gives

```
----    266 PARAMETER SUMMARY       Farm Summary
            Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resourc
Corn                                     20.00      200.00
Soybeans                                480.00
Beef
cattle                                                          615.79
Water                                                                       16
Cropland                    700.00                                          128
Pastureland                 130.00                                           84
Cattlefeed
Total         162685.05                 500.00      200.00
```

while a display arising from line 359 which is associated with the $0.70 beef price gives

```
----    359 PARAMETER SUMMARY       Farm Summary
            Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resourc
Corn                                     22.84      160.85
Soybeans                                489.86
Beef
cattle                                                          866.67
Cropland                    673.55
Pastureland                 130.00                                         1450
Cattlefeed
Total         373686.10                 512.70      160.85
```

and the display arising from line 431 which is associated with the $2.70 corn price gives

```
----    431 PARAMETER SUMMARY       Farm Summary
            Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resourc
Corn                                     31.98      200.00
Soybeans                                410.24
Beef
cattle                                                          866.67
Water                                                                       15
Cropland                    642.22
Pastureland                 130.00                                         1310
Cattlefeed
Total         375839.30                 442.22      200.00
```

Now let's appraise how good this is. In my judgment it is not so good for four reasons

© 2022 Prof. Bruce McCarl

- The output is inconveniently spread over 250 lines in the output.
- No cross scenario comparison is present.
- The beef price is troublesome because after I raised it to $0.70 it remains there is the third scenario.
- There is a lot of repetition in handling of solves and report writing.

Lets fix these up.

8.2.1 Introducing cross scenario report writing

First suppose we gather the output into one spot. We can do this by introducing a couple of new sets and a parameter (**mancompb.gms**). The set

```
set scenarios /base,beefp,beefcorn/;
set ordr /"Scenario setup","Scenario Results"/;
```

identifies the scenarios and sets up a place to save the assumptions and the results. In turn the new parameter adds two dimensions to the summary parameter and retains it.

```
Parameter savsumm(ordr,*,alli,scenarios) Comparative Firm Summary;
savsumm("Scenario setup","price",primary,"base")=price(primary);
savsumm("Scenario Results",measures,alli,"base")=summary(alli,measures);
```

where the Scenario setup line copies in the current setup of the price vector and the Scenario Results line copy in the results as stored in the summary parameter.

In turn our code is

```
$include farmcomp.gms
$include farmrep.gms
set ordr /"Scenario setup","Scenario Results"/
set scenarios /base,beefp,beefcorn/
Parameter savsumm(ordr,*,alli,scenarios) Comparative Farm Summary;
savsumm("Scenario setup","price",primary,"base")=price(primary);
savsumm("Scenario Results",measures,alli,"base")=summary(alli,measures);
price("beef")=0.70;
SOLVE Firm USING LP MAXIMIZING NETINCOME;
display price ;
$include Farmrep.gms
savsumm("Scenario setup","price",primary,"beefp")=price(primary);
savsumm("Scenario Results",measures,alli,"beefp")=summary(alli,measures);
price("corn")=2.70;
display price ;
SOLVE Firm USING LP MAXIMIZING NETINCOME;
$include Farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn")=price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn")=summary(alli,measures);
```

© 2022 Prof. Bruce McCarl

```
option savsumm:2:3:1;display savsumm;
```

The results are

```
----    439 PARAMETER savsumm  Comparative Farm Summary

                                                      base       beefp    beefcorn
Scenario setup  .price         .Corn                  2.20        2.20        2.70
Scenario setup  .price         .Soybeans              5.00        5.00        5.00
Scenario setup  .price         .Beef                  0.50        0.70        0.70
Scenario Results.Net Income    .Total            162685.05   373686.10   375839.30
Scenario Results.Land use      .Cropland            700.00      673.55      642.22
Scenario Results.Land use      .Pastureland         130.00      130.00      130.00
Scenario Results.Dry Cropping  .Corn                 20.00       22.84       31.98
Scenario Results.Dry Cropping  .Soybeans            480.00      489.86      410.24
Scenario Results.Dry Cropping  .Total               500.00      512.70      442.22
Scenario Results.Irr Cropping  .Corn                200.00      160.85      200.00
Scenario Results.Irr Cropping  .Total               200.00      160.85      200.00
Scenario Results.Livestock     .cattle              615.79      866.67      866.67
Scenario Results.Resource Value.Water               16.83                   15.99
Scenario Results.Resource Value.Cropland            128.49
Scenario Results.Resource Value.Pastureland          84.26     1456.90     1316.09
Scenario Results.Resource Value.April Labor          32.34       82.29       61.39
Scenario Results.Resource Value.May Labor            27.01       80.53       61.72
Scenario Results.Resource Value.Sept Labor                       53.57       84.92
Scenario Results.Resource Value.Oct Labor            11.50       46.21       87.21
Scenario Results.Product Value .Corn                  2.20        2.34        2.70
Scenario Results.Product Value .Soybeans              5.00        5.00        5.00
Scenario Results.Product Value .Beef                  0.50        0.70        0.70
Scenario Results.Product Value .Cattlefeed            4.71        4.89        5.36
```

where we have now achieved a cross scenario report and place all the output in one place.

8.2.1.1 Percentage change cross scenario reports

However we may still wish some percentage change calculations which we achieve by adding

```
Parameter savsummp(ordr,*,alli,scenarios) Comparative Farm Summary (percent chg)
savsummp(ordr,measures,alli,scenarios)$savsumm(ordr,measures,alli,"base")=
        round(  (savsumm(ordr,measures,alli,scenarios)
          -savsumm(ordr,measures,alli,"base"))*100
          /savsumm(ordr,measures,alli,"base"),1);
savsummp(ordr,measures,alli,scenarios)
          $(savsumm(ordr,measures,alli,"base") eq 0
        and savsumm(ordr,measures,alli,scenarios) ne 0)=na;
option savsummp:1:3:1;display savsummp;
```

which sets up a new parameter, computes percentage changes rounded to one place and sets them to

© 2022 Prof. Bruce McCarl

na if the base number is zero yielding

```
----    456 PARAMETER savsummp  Comparative Farm Summary (percent chg)

                                                        beefp    beefcorn

Scenario setup  .price          .Corn                                22.7
Scenario setup  .price          .Beef                 40.0           40.0
Scenario Results.Net Income     .Total               129.7          131.0
Scenario Results.Land use       .Cropland             -3.8           -8.3
Scenario Results.Dry Cropping   .Corn                 14.2           59.9
Scenario Results.Dry Cropping   .Soybeans              2.1          -14.5
Scenario Results.Dry Cropping   .Total                 2.5          -11.6
Scenario Results.Irr Cropping   .Corn                -19.6
Scenario Results.Irr Cropping   .Total               -19.6
Scenario Results.Livestock      .cattle              40.7           40.7
Scenario Results.Resource Value.Water              -100.0           -5.0
Scenario Results.Resource Value.Cropland           -100.0         -100.0
Scenario Results.Resource Value.Pastureland        1629.0         1461.9
Scenario Results.Resource Value.April Labor         154.4           89.8
Scenario Results.Resource Value.May Labor           198.1          128.5
Scenario Results.Resource Value.Sept Labor             NA             NA
Scenario Results.Resource Value.Oct Labor           301.8          658.4
Scenario Results.Product Value .Corn                  6.4           22.7
Scenario Results.Product Value .Beef                 40.0           40.0
Scenario Results.Product Value .Cattlefeed            3.9           13.8
```

8.2.2 Preserving changed data

In the above example the management of the beef price was questionable. In particular the beefprice is "static", as discussed in the chapter Calculating Items, and since GAMS lives for the moment, when a calculation is issued then all prior values are overwritten for calculated items and that value is retained from then on. In this case the "repeated static" buildup and to manage thus we need to reset the data. We do this by first saving then reestablishing the price vector before each change as is done in **mancompc.gms**

```
$include farmcomp.gms
$include farmrep.gms
parameter saveprice(alli) saved prices;
saveprice(alli)=price(alli);
set ordr /"Scenario setup","Scenario Results"/
set scenarios /base,beefp,beefcorn/
Parameter savsumm(ordr,*,alli,scenarios) Comparative Farm Summary;
savsumm("Scenario setup","price",primary,"base")=price(primary);
savsumm("Scenario Results",measures,alli,"base")=summary(alli,measures);
price(alli)=saveprice(alli);
price("beef")=0.70;
SOLVE FARM USING LP MAXIMIZING NETINCOME;
display price ;
```

© 2022 Prof. Bruce McCarl

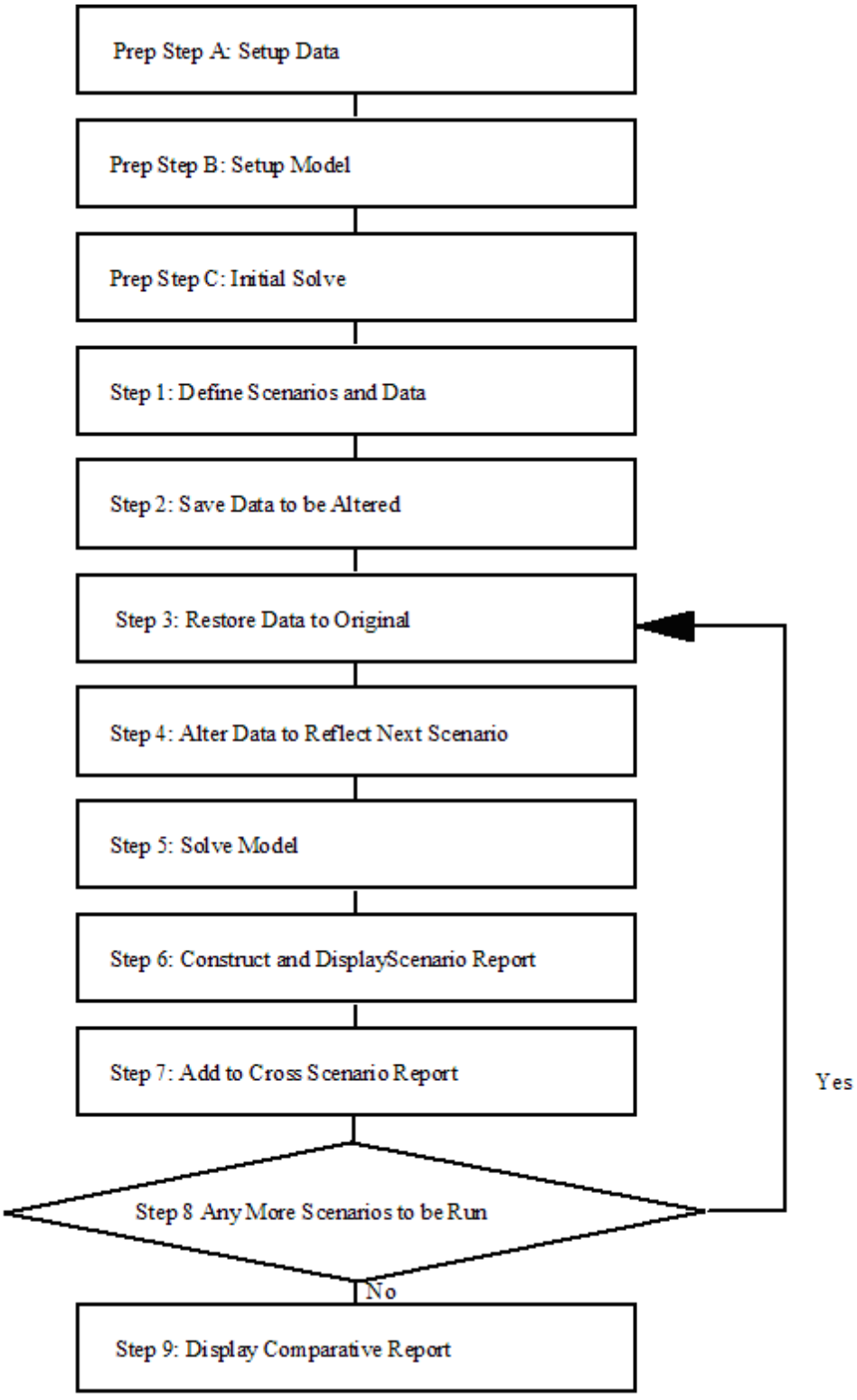
```
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefp")=price(primary);
savsumm("Scenario Results",measures,alli,"beefp")=summary(alli,measures);
price(alli)=saveprice(alli);
price("corn")=2.70;
display price ;
SOLVE FARM USING LP MAXIMIZING NETINCOME;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn")=price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn")=summary(alli,measures);
option savsumm:2:3:1;display savsumm;
```

which resets the data to base levels before each scenario.

8.3 An automated approach - avoiding repeated work

The basic structure of a comparative analysis is outlined in Figure 1. The first three boxes reflect preparatory steps that would be done in a conventional GAMS program where one sets up the initial data and model then solves. In the box labeled step 1, the comparative model analysis begins. There the scenarios are identified and the scenario data defined. After that I save the data that are to be changed during the scenario runs preserving Abase@ scenario values. I then enter a loop that is repeated for each scenario to be analyzed. In that loop the first task is to restore the data to its base scenario levels (step 3). This is done so that I always start from the same data. For example, if I am altering prices in some scenarios and costs in others once I changed the prices they will be altered forever unless they are changed back (restored) to their original values. Then the data and model differences for that scenario are imposed in step 4 and the model solved in Step 5. Step 6 involves a report on the individual scenario with items displayed as desired. Then in step 7 parameters for cross scenario comparative reports are saved. In step 8 I check to see if more scenarios are to be solved and if so return to repeat steps 3-8 until all scenarios are completed. Finally, I display a comparative report that presents the information saved across scenarios.

© 2022 Prof. Bruce McCarl



This is implemented in the file **compare.gms**

© 2022 Prof. Bruce McCarl

```
*step 1 - setup scenarios
set ordr /"Scenario setup","Scenario Results"/
set scenarios /base,beefp,beefcorn/
Parameter savsumm(ordr,*,alli,scenarios) Comparative Farm Summary;
table scenprice(primary,scenarios)  price alterations by scenario
                base    beefp   beefcorn
corn                                2.70
soybeans
beef                    0.70            ;
*step 2 save data
parameter savprice(primary) saved primary commodity prices;
savprice(primary)=price(primary);
loop(scenarios,
*step 3 reestablish data to base level
    price(primary)=savprice(primary);
*step 4 change data to levels needed in scenario
    price(primary)$scenprice(primary,scenarios)=scenprice(primary,scenarios);
    display price;
*step 5 -- solve model
   SOLVE FARM USING LP MAXIMIZING NETINCOME;
*step 6 single scenario report writing
$include farmrep.gms
*step 7 cross scenario report writing
    savsumm("Scenario setup","price",primary,scenarios)=price(primary);
    savsumm("Scenario Results",measures,alli,scenarios)=summary(alli,measures);
*step 8 end of loop
    );
*step 9 compute and display final results
option savsumm:2:3:1;display savsumm;
```

The above comparative analysis code contains a LOOP statement as discussed in the Control Structures chapter. The LOOP statement tells GAMS to repeat execution of all the statements enclosed in the parentheses defining the LOOP. In this case GAMS will repeat execution of the statements starting with the assignment of

```
price(primary)=savprice(primary);
```

through

```
savsumm("Scenario Res",measures,alli,scenarios)     =summary(alli,measures);
```

for each element of the scenarios set avoiding the repetition.

Note in the loop you can address the scenarios set and only the one element active at that stage of the loop will be considered.

Adding a scenario
Changing model structure

© 2022 Prof. Bruce McCarl

8.3.1 Adding a scenario

Now let me illustrate the full extent to which the above approach is automated by adding a new scenario. This is done through two modifications. First we expand the set of named scenarios to include the new one. Second we add data for the new scenario (**comparenew.gms**).

```
set scenarios /base,beefp,beefcorn,new/
table scenprice(primary,scenarios)  price alterations by scenario
                base    beefp   beefcorn  new
corn                                2.70
soybeans                                  4.32
beef                    0.70                ;
```

The rest of the code is unchanged. Note we did not have to add a solve or report writing or anything else. That is all handled by the LOOP and one just has to add data to add to the scenarios.

In doing such an analysis cases certainly arise where a new scenario requires that other data items like resource endowments be changed. In such a case, and one needs to alter the code so that with respect to the other data item the code saves, resets to base levels and alters to reflect the scenario. For example if our new scenario also involved alteration of the data item available in the cropland category making 30% more available, we could add code like the following (**compareother.gms**)

```
table scenavailable(alli,scenarios)  price alterations by scenario
                base    beefp   beefcorn  new
cropland                                  1.3;
*step 2 save data
parameter savprice(primary) saved primary commodity prices;
savprice(primary)=price(primary);
parameter saveavailable (alli);
saveavailable (alli)= available (alli);
loop(scenarios,
*step 3 reestablish data to base level
    price(primary)=savprice(primary);
    available (alli)=saveavailable (alli);
*step 4 change data to levels needed in scenario
    price(primary)$scenprice(primary,scenarios)=scenprice(primary,scenarios);
    available(alli)$scenavailable(alli,scenarios)=
       available(alli)*scenavailable(alli,scenarios);
    display price,available;
```

8.3.2 Changing model structure

Many studies involve model structure modifications. A context sensitive model structure by making constraints or terms in the model conditional (see the discussion in the Conditionals chapter). In particular in the example a conditional constraint on maximum cattle (cowlimit) could be set up based on a scalar (cowlim) and controlled by the scenario looping procedure (**comparemod.gms**).

```
scalar cowlim activates cowlimit constraint /1/
equation cowlimit conditional equation on cow limits;
```

© 2022 Prof. Bruce McCarl

```
cowlimit$cowlim.. sum((animals,livemanage),
            liveprod(animals,livemanage))=l=100;
model farmcowlim /all/
parameter cowlims(scenarios) cowlimit by scenario
          /base 0, cowlim 1/;
loop(scenarios,
    cowlim=cowlims(scenarios);
    SOLVE FARMcowlim USING LP MAXIMIZING NETINCOME;
);
```

which imposes the constraint in a conditional manner.

It can be desirable from the standpoint of integrity of an advanced basis (see discussion in the Advanced Basis Usage chapter) to make the model size invariant. In such a case one would alter the code above so the constraint was practically rather than physically removes. This could be accomplished by altering the code above to be

```
scalar cowlim activates cowlimit constraint /1/
equation cowlimit conditional equation on cow limits;
cowlimit.. sum((animals,livemanage),
      liveprod(animals,livemanage))=l=100+(1-cowlim)*1000000;
model farmcowlim /all/
parameter cowlims(scenarios) cowlimit by scenario
          /base 0, cowlim 1/;
loop(scenarios,
    cowlim=cowlims(scenarios);
    SOLVE FARMcowlim USING LP MAXIMIZING NETINCOME;
);
```

which keeps the constraint but limits the farm to the unattainable million+ cows if the unlimited scenario is being run. Similarly for variables to be eliminated one could alter objective function coefficients so they became very undesirable (adding a term to a maximization problem like (3-1000000$eliminatevar)*x to the objective equation).

8.4 Where am I?

Suppose you are running a comparative analysis job which takes a long time with many solves and you want to know the stage it is on. You can get such information to the screen by including the following pieces GAMS code.

To define where output is to be sent for messages to the screen and the log file (**comparewhere.gms**).

```
$set console
$if %system.filesys% == UNIX  $set console /dev/tty
$if %system.filesys% == DOS $set console con
$if %system.filesys% == MS95  $set console con
$if %system.filesys% == MSNT  $set console con
$if "%console%." == "." abort "filesys not recognized";
file screen / '%console%' /;
```

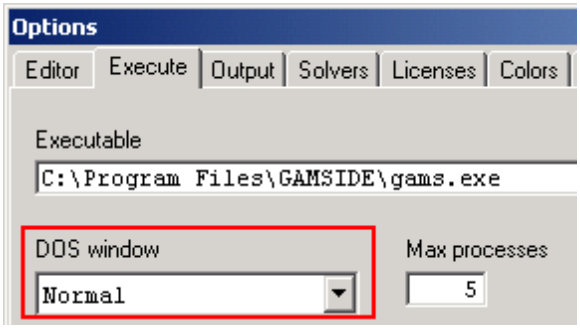
© 2022 Prof. Bruce McCarl

```
file log /''/
```

To send the output to the screen and the log (IDE process window)

```
loop(scenarios,
    put screen;
    put 'I am on scenario ' Scenarios.tl;
    putclose;
    put log;
    put 'I am on scenario ' Scenarios.tl;
    putclose;
```

The screen part works fine in DOS or UNIX but not in IDE (there you need to make the DOS window visible by manipulating the options under the execute tab or just send to the LOG file).



The LOG part put entries in the IDE process window.

One can also change the name of the DOS box in windows versions starting with Windows 2000 using the put file sequence (**dosbox.gms**)

```
File dosbox / titlemaker.cmd/
putclose dosbox '@title this is a new DOS box title';
execute 'titlemaker.cmd'
```

where again the DOS window must be visible if this is used in the IDE.

9 GAMS Command Line Parameters

When GAMS is called one can use a number of arguments on the command line or in the GAMS parameters box in the upper right hand corner of the IDE screen. Here I discuss the universe of available command line parameters.

Important parameters
Alphabetic list of all GAMS command line parameters

© 2022 Prof. Bruce McCarl

9.1 Important parameters

The command line parameters break into classes as listed below. Also note one can employ these command line options in a text file to permanently customize all GAMS runs done on a computer as discussed in the Customizing GAMS chapter. Here we list the parameters divided by category and below provide a composite alphabetic list.

Compiler function – regional settings
Error detection and correction
LST and LOG output content and format control
Solver name choice
Option file presence
Directory management
Saves and restarts
User defined options

9.1.1 Compiler function – regional settings

One can change command line parameters and in turn cause GAMS to alter the way the compiler functions allowing the user GMS files to allow inclusion of additional characters and modify the time date formats of output using the options below. Such changes are often desirable in an international setting.

Parameter	Brief description
Charset	Command line parameter that controls compiler use of extended character set including European and other international characters.
Dformat	Command line parameter that controls the date format and allows European date conventions.
Tformat	Command line parameter that controls time format.

9.1.2 Error detection and correction

A few options are available that influence error message placement, error condition correction and monitoring of statement execution performance characteristics.

Parameter	Brief description
Cerr	Command line parameter that controls the compile time error limit.
Errmsg	Command line parameter that controls the position of error messages in the echo print and through use of ERRMSG=2 allows one to reposition error messages to just after error marking (Standard Output).
Errnam	Command line parameter that specifies the name of a file defining the internally used compiler error messages.
Error	Command line parameter that forces a parameter error with a specified message.
Execerr	Command line parameter that defines a limit on the number of errors that can be found during execution or preprocessing associated with a solve statement.
Ferr	Command line parameter that controls redirection of compilation error

© 2022 Prof. Bruce McCarl

	messages to a file.
Profile	Command line parameter that causes GAMS to include information on statement execution time and memory use in LST file allowing one to find slow or large memory using statements (Speed, Memory).

9.1.3 LST and LOG output content and format control

A group of options allow some control of the contents of the LOG and LST output files. These are mostly discussed in Standard Output.

Parameter	Brief description
Appendlog	Command line parameter that controls whether the program overwrites or appends to the LOG file.
Appendout	Command line parameter that controls whether the program overwrites or appends to the LST file.
Case	Command line parameter that controls the case of the text in the LST file for the echo print.
Dumpparms	Command line parameter that provides more detailed information about the parameters changed or set by the user, GAMS or the IDE during the current run.
DumpParmsLogPrefix	Command line parameter causes log lines that are triggered by DumpParms=2 to be prefixed with a specific string for easy filtering of these lines
Errorlog	Command line parameter that controls the error message lines written to the LOG file.
Limcol	Command line parameter that includes the first n cases for each named variable in the LST file.
Limrow	Command line parameter that includes the first n cases for each named equation in the LST file.
Logfile	Command line parameter that specifies the name of the LOG file.
Logline	Command line parameter that controls the amount of line tracing to the LOG file.
Logoption	Command line parameter controls destination for the LOG file, used with setting of 0 or 2 to permit UNIX jobs to operate in the background.
Output	Command line parameter that specifies the name of file containing the output (LST file).
Pagecontr	Command line parameter that affects the page control in the LST file.
Pagesize	Command line parameter that sets the default number of lines per page. If less than 30 it will be reset to the default of 9999.
Pagewidth	Command line parameter that sets the default number of columns on a page. This value should be between 72 and 32767.

© 2022 Prof. Bruce McCarl

Profile	Command line parameter that tells GAMS whether to include statement execution time and memory use reporting to the LST file.
Profiletol	Command line parameter that specifies the minimum time a statement must use to appear in the profile generated output.
Solprint	Command line parameter that controls the printing of the solution report.
Stepsum	Command line parameter that controls the generation of a step summary of the processing times taken by GAMS during a given run.
Suppress	Command line parameter that tells GAMS whether to suppress the compiler echo print of the GAMS input instructions.
Sysout	Command line parameter that controls the incorporation of additional solver generated output to the LST file.
Tabin	Command line parameter that tells GAMS how to deal with tabs
Timer	Command line parameter that specifies an instruction timer threshold in milli seconds.

9.1.4 Solver name choice

A group of options allow one to make a command line specification of which solver is to be used.

Parameter	Brief description
CNS	Command line parameter that gives the name of the CNS model solver.
DNLP	Command line parameter that gives the name of the DNLP model solver.
EMP	Command line parameter that gives the name of the EMP model solver.
LP	Command line parameter that gives the name of the LP model solver.
MCP	Command line parameter that gives the name of the MCP model solver.
MINLP	Command line parameter that gives the name of the MINLP model solver.
MIP	Command line parameter that gives the name of the MIP model solver.
MIQCP	Command line parameter that gives the name of the MIQCP model solver.
MPEC	Command line parameter that gives the name of the MPEC model solver.
NLP	Command line parameter that gives the name of the NLP model solver.
RMINLP	Command line parameter that gives the name of the RMINLP model solver.
RMIP	Command line parameter that gives the name of the RMIP model solver.
QCP	Command line parameter that gives the name of the QCP model solver.
RMIQCP	Command line parameter that gives the name of the RMIQCP model

© 2022 Prof. Bruce McCarl

	solver.
RMPEC	Command line parameter that gives the name of the RMPEC model solver.

9.1.5 Option file presence

One may use options to change the location and default existence of the solver option file rather than having to use the command modelname.optfile=1 in each and every GMS file.

Parameter	Brief description
Optdir	Command line parameter that gives the name of the directory to be used by GAMS for solver option files.
Optfile	Command line parameter that gives the number to use by default for model.optfile.

9.1.6 Directory management

One can redefine both the search tree for GAMS input files, as well as relocating places where libinclude files will be drawn from and other file locations.

Directory Parameter	Brief description
Curdir	Command line parameter that gives the name of the working directory.
Griddir	Command line parameter that gives the name of the grid file directory.
Inputdir	Command line parameter that gives the search path for files.
Inputdir1 to inputdir40	Command line parameters that give the names of the individual directories to be searched.
Libincdir	Command line parameter that gives the name of the directory where libinclude files are kept.
Maxprocdir	Command line parameter that gives the maximum number of workfile directories that can be generated by GAMS.
Optdir	Command line parameter that gives the name of the directory to be used by GAMS for solver option files.
Procdir	Command line parameter that gives the name of the directory where the work files generated by GAMS will be stored.
Putdir	Command line parameter that gives the name of the directory where files generated by the put command will be stored.
Scrdir	Command line parameter that gives the name of the directory to be used by GAMS for temporary files generated during execution.
Sysdir	Command line parameter that gives the name of the directory where GAMS executables reside.
Sysincdir	Command line parameter that gives the name of the directory where sysinclude files are kept.
Workdir	Command line parameter that gives the name of the working directory.

File Parameter	Brief description

© 2022 Prof. Bruce McCarl

GDX	Command line parameter that gives the name of GAMS data exchange file to write at program end.
License	Command line parameter that points to the GAMS license file

9.1.7 Saves and restarts

Options control whether or not save and restart files will be used (Save Restart). They also permit a platform independent restart file to be written that can be moved between computers with different operating systems.

Parameter	Brief description
Fsave	Command line parameter that forces GAMS to write a saved workfile even in the face of execution or other errors.
Restart	Command line parameter that gives the name of restart file.
Save	Command line parameter that gives the name of save file.
Xsave	Command line parameter that specifies the name of a save file written in ASCII format so it is platform independent. Note restart automatically will read this file type.

One can change the names of the items in a save restart file using SymPrefix although one cannot start a program with 2 restart files. Thus one may need to do a restart with the changed names then save items to a gdx file and eventually load in the desired stuff elsewhere.

Users can work with models where it is desirable to solve models on computational grids or clouds outside a secure computing environment. GAMS contains facilities that allows one to prepare a restart files that within it has has all the names and other documentation related to a specific model run changed before moving it to the less secure environment. This is duiscussed under the topic of obfuscated save and restarts.

GAMS also contains ways to take a subsequent save file containing results and back translate it inside the secure environment.

This is done using an "obfuscated" save files, which is a save files where symbol and UEL names have been obfuscated meaning all the names have been changed to computer generated labels that do not convey problem information. This is done to allow one to enhance the security of information in the saved files particularly if operating in a cloud computing environment.

To do this one employs the command line options `saveobfuscate` (abbreviated as `so`) and `xsaveobfuscate` (abbreviated as `xso`) that respectively generate uncompressed and compressed obfuscated save files. Also the command line option `restartNamed` (`rn`) can be used to bring back the original names when restarting from an obfuscated save file.

The syntax for use is as follows.

© 2022 Prof. Bruce McCarl

- To run a GAMS model into a named and an obfuscated save file:

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

- To bring the obfuscated save file in and do a continued compilation with reporting and export one would use

```
gams more r=0anon restartNamed=0named
```

The obfuscated work file is basically the original named work file but with all strings for symbols , set elements and explanatory text replaced with obfuscated names. GAMS obfuscates by keeping the original length of the symbol/label and creating a sequence of strings e.g. for symbols of length 3 we create A00, A01, A02, ..., Z . Similarly for explanatory text, GAMS always uses the single quote character and creates many weird looking labels. For the other strings (symbol text and label text) we just change all characters in these strings. The only other strings obfuscated are the titles and subtitles. File names are not obfuscated.

The basic way this works is on the original machine you run

```
gams mymodel a=c s=0named so=0obfuscated
```

which yields an obfuscated restart file plus a local restart file that gives instructions for back translating the information. Note these files have not been executed since a compile only command lime parameter (a=c) was used

Now we move the obfuscated work file to a non-secure machine and start up execution of an empty file there restarting from the obfuscated save file and the in turn save the results.
The empty file can be created using

```
execute "echo * Empty > jobforremote.gms"
```

the run can then be done as

```
gams jobforremote r=0obfuscated s=1obfuscated
```

which will execute the original job on the remote machine but in an obfuscated manner and create a saved file with the results.

In turn one can bring that file back to the original machine and run an additional job translating the results back using the local save file

```
gams continue r=1obfuscated rn=0named
```

9.1.8 More Secure "obfuscated" saves and restarts

Users can find it desirable to solve models on computational grids or clouds that are outside a secure computing environment. GAMS contains facilities that allows one to prepare a restart files that within it alters all the names and other documentation related to the model so it is virtually impossible to figure out the context. In turn, then this can be more securely used in the less secure environment.

GAMS also contains ways to take a subsequent save file containing results from the less secure

© 2022 Prof. Bruce McCarl

environment and back translate it inside the original secure environment.

This is done using an "obfuscated" save files, which is a save file where symbol and UEL names have been obfuscated. Generally this means all the text strings have been changed to computer generated labels that do not convey problem information.

To do this one employs the command line options saveobfuscate (abbreviated as so) and xsaveobfuscate (abbreviated as xso) that respectively generate uncompressed and compressed obfuscated save files. Also the command line option restartNamed (rn) can be used to bring back the original names when restarting from both an obfuscated save file and a regular save file.

The syntax for use is as follows.

- To save a GAMS model for future execution into an obfuscated save file one does

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

  where one also must create and retain a regular save file so back translation can be done

- To start from the obfuscated save file and do a continued job with reporting and whatever else one would use

```
gams more r=0anon rn=0named
```

The obfuscated work file is basically the original save file but with all strings for symbols , set elements and explanatory text replaced with obfuscated names.

GAMS **obfuscates** by keeping the original length of the symbol/label and creating a sequence of strings e.g. for symbols of length 3 we create A00, A01, A02, ..., Z . Similarly for explanatory text, GAMS always uses the single quote character and creates many weird looking labels. For the other strings (symbol text and label text) we just change all characters in these strings. The only other strings obfuscated are the titles and subtitles. File names are not obfuscated.

The basic way this works is on the original machine you run

```
gams mymodel a=c s=0named so=0obfuscated
```

which yields an obfuscated restart file plus a local restart file that gives instructions for back translating the information. Note these files have not been executed since a compile only command lime parameter (a=c) was used

Now we move the obfuscated work file to a non-secure machine and start up execution of an empty file there restarting from the obfuscated save file and the in turn save the results.
The empty file can be created using

```
execute "echo * Empty > jobforremote.gms"
```

© 2022 Prof. Bruce McCarl

the run can then be done as

```
gams jobforremote r=0obfuscated s=1obfuscated
```

which will execute the original job on the remote machine but in an obfuscated manner and create a saved file with the results.

In turn one can bring that file back to the original machine and run an additional job translating the results back using the local save file

```
gams continue r=1obfuscated rn=0named
```

9.1.9 User defined options

A set of options allow users to pass control variables and text strings of their own definition into the program for their use. A set of text strings may be passed through 5 user options that are not used by GAMS but are entirely under user control and require definitions in the GAMS code to implement their use (Conditional Compile).

Parameter	Brief description
--	Command line parameter that allows definition of a control variable.
//	Command line parameter that allows definition of a control variable.
/-	Command line parameter that allows definition of a control variable.
-/	Command line parameter that allows definition of a control variable.
Integer 1 to Integer5	Command line parameter that can contain any integer number.
User1 to user5	Command line parameter that permits entry of text for up to 5 user defined options.

9.2 Alphabetic list of all GAMS command line parameters

Here I list all options that can be used on a GAMS command line or included in the GAMSPARM.txt file to customize default values.

Each of the commands below is implemented in one of three ways

- In the GAMS command line using the syntax

```
GAMS filename command=stringornumber
```

- In the command line box of the IDE or in its default execution parameters (Gamside)
- In the gmsprmxx.txt file to customize GAMS function of a computer.

-- // -/ /-- on command lines
Action: A
Integer1 to Integer5
IntVarup
RMIQCP
RMPEC

© 2022 Prof. Bruce McCarl

Appendexpand: Ae
Appendlog: Al
Appendout: Ao
Bratio
Case
Cerr
Charset
CNS
Curdir: Cdir
Dformat: Df
DNLP
Domlim
Dumpopt
Dumpparms: Dp
DumpparmsLogPrefix:Dplp
EMP
Eolonly: Ey
Errmsg
Errnam
Error
Errorlog: Er
Etlim
Execerr
Execmode
Expand: Ef
FDDelta
FDopt
Ferr
Filecase
Filestem
Forcework: Fw
Forlim
Fsave
G205
Gdx
Gdxcompress
Gdxconvert
Gdxuels
Griddir: Gdir
Gridscript: Gscript
Heaplimit: Hl
Holdfixed
Ide
Input: I
Inputdir: Idir
Inputdir1 to inputdir40:
Idir1 to idir40
Iterlim
Jobtrace: Jt
Keep
Libincdir: Ldir
License
Limcol
Limrow
Logfile: Lf
Logline: Ll
Logoption Lo
LP
Maxprocdir
MCP
Mcprholdfx
MINLP
MIP
MIQCP
MPEC
Multipass: Mp
NLP
Nodlim
NoNewVarEqu
On115
Optca
Optcr
Optdir
Optfile
Output: O
Pagecontr: Pc
Pagesize: Ps
Pagewidth: Pw
Parmfile: Pf
Plicense
Procdir
Profile
Profilefile
Profiletol: Ptol
Putdir: Pdir
QCP
Reference: Rf
Reslim
Restart: R
RMINLP
RMIP
Save: S
Savepoint: Sp
Scrdir: Sd
ScrexT: Se
Scriptexit
Scriptfrst: Sf
Scriptnext: Script
Scrnam: Sn
Seed
Solprint
Solvelink
Solver
Solvercntr: Scntr
Solverdict: Sdict
Solverinst: Sinst
Solvermatr: Smatr
Solversolu: Ssolu
Solverstat: Sstat
Stepsum
Stringchk
Subsys
Suppress
Symbol
SymPrefix
Sys10
Sys11
Sysdir
Sysincdir: Sdir
Sysout
Tabin
Tformat: Tf
Threads
ThreadsAsync
Timer
Trace
Traceopt
User1 to user5: U1 to U5
Warnings
Workdir: Wdir
Workfactor
Workspace
Xsave: Xs
Zerores
Zeroresrep

© 2022 Prof. Bruce McCarl

9.2.1 -- // -/ /-- on command lines

A set of 2 minus signs followed by a name defines a control variable as discussed in the conditional compile chapter. The symbol strings // , -/ , and /- all do the same.

The syntax is any of the following four

```
--name=string
//name=string
/-name=string
-/name=string
```

where name is the name of a control variable chosen by the user and string a text string.

For example (**minusminus.gms**) one could use

```
--keycity=Boston //myflag=modes /-myvalue=7.6 -/dothis="display x;"
```

specifying the control variables keycity, myflag, myvalue and dothis with the strings Boston, modes, 7.6 and display x; respectively. In tune when one had a statement

```
x("%keycity%","%myflag%")=%myvalue%;
%dothis%
```

it would become

```
x("boston","mode")=7.6;
display x;
```

9.2.2 Action: A

This keyword controls the type of compiling action that will be undertaken by GAMS. String input is expected with the allowable string values being

CE	compile and execute (default)
C	compile only
E	execute only
G	glue code generation
R	restart after a solve

The command is implemented with either

```
Action=string
```

or

```
A=string
```

where string is one of the choices above. This command can sometimes be useful when one wishes to compile but not execute.

© 2022 Prof. Bruce McCarl

9.2.3 Appendexpand: Ae

This keyword controls the manner of file opening of the expand file. Numeric input is expected with the allowable numeric values being

0	open with rewrite
1	open with an append (default)

The command is implemented with either

```
Appendexpand=number
```

or

```
Ae=number
```

9.2.4 Appendlog: Al

This keyword controls whether the program overwrites or appends to the LOG file. Numeric input is expected with the allowable numeric values being

0	overwrite LOG file (default)
1	append to LOG file

The command is implemented with either

```
Appendlog=number
```

or

```
Al=number
```

9.2.5 Appendout: Ao

This keyword controls whether the program overwrites or appends to the LST file. Numeric input is expected in with the allowable numeric values being

0	overwrite LST file (default)
1	append to LST file

The command is implemented with either

```
Appendout=number
```

or

```
Ao=number
```

9.2.6 Bratio

This option specifies what GAMS will do in forming an advanced basis as discussed in the Basis

© 2022 Prof. Bruce McCarl

chapter. This option is used by setting

```
Option Bratio=realnumber;
```

The value specified for this option causes a basis to be discarded if the number of basic variables is smaller than bratio times the number of equations.

Notes:

- Setting bratio to 1 will always cause the basis to be discarded, which is sometimes needed with nonlinear problems as discussed in the NLP and Basis chapters.
- Setting bratio to 0 forces GAMS to always try to construct a basis.
- 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.
- The allowable values range from 0 to 1 with a default value of 0.25.

9.2.7 Case

This keyword controls the case of the text in the LST file for the echo print. Numeric input is expected with the allowable numeric values being

`0`	write LST file GAMS statement echo print in mixed case (default)
`1`	write GAMS statement echo print file in upper case only

The command is implemented with either

```
Case=number
```

9.2.8 Cerr

This keyword controls the compile time error limit. The compilation will be stopped after n errors have occurred. Numeric input is expected with the allowable numeric values being

`0`	no error limit (default)
`n`	stop after n errors

The command is implemented with

```
Cerr=number
```

9.2.9 Charset

This keyword controls compiler use of the extended character set which permits the compiler to accept international characters in comments and text items (European and other characters). Numeric input is

© 2022 Prof. Bruce McCarl

expected with the allowable numeric values being

```
0       use limited GAMS characters set
1       accept any character in comments and other text entries (default)
```

The command is implemented with

```
Charset=number
```

9.2.10 CNS

This keyword controls the name of the solver that will be used to solve CNS models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
CNS=solvername
```

9.2.11 Curdir: Cdir

This keyword gives the name of the current working directory. If not specified, it will be set to the directory the GAMS module is called from. String input is expected.

The command is implemented with

```
Curdir=directoryname
```

or

```
Cdir= directoryname
```

9.2.12 Dformat: Df

This keyword controls compiler use of alternative date formats. Numeric input is expected with the allowable numeric values being

```
0       mm/dd/yy   (month/day/year) (default)
1       dd.mm.yy
2       yy-mm-dd
```

The command is implemented with

```
Dformat=number
```

or

```
Df=number
```

© 2022 Prof. Bruce McCarl

9.2.13 DNLP

This keyword controls the name of the solver that will be used to solve DNLP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
DNLP=solvername
```

9.2.14 Domlim

This option sets the maximum number of domain errors. It allows errors to occur up to the given number during solution.Numeric input is expected with the allowable numeric values being

0	allows 0 errors to occur (default)
n	allows n errors to occur

The command is implemented with

```
Domlim=number
```

9.2.15 Dumpopt

This keyword 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. Numeric input is expected with the allowable numeric values being

0	no dump file (default)
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

The command is implemented with

```
Dumpopt=number
```

Example:

To illustrate the use of the dumpopt option, [TRNSPORT] 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 / ;
```

© 2022 Prof. Bruce McCarl

```
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/ ;
```

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 [TRNSPORT] 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 [TRNSPORT].
To illustrate the use of the dumpopt option, run the second model using the following command

© 2022 Prof. Bruce McCarl

```
gams trans2 restart=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;
```

© 2022 Prof. Bruce McCarl

```
*      *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
```

9.2.16 Dumpparms: Dp

This keyword lists the settings of all command line parameters changed or set by the user, GAMS or the IDE during the current run. Numeric input is expected with the allowable numeric values being

0	no logging (default)
1	lists altered parameters in the log file
2	creates a list of altered parameters in the log file with the line marked with ***

The command is implemented with

```
Dumpparms=number
```

or

```
Dp=number
```

For example using the command line parameter dp=2 on the example model gdxintrnsportinfersets.gms results in a log file that contains the following

```
*** opennotype R=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.gms"
*** opentext    W=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.lxi"
*** opentext    W=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.lst"
***     LP CPLEX
***     MIP CPLEX
***     RMIP CPLEX
***     MINLP DICOPT
***     MIQCP DICOPT
***     Input C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\gdxintrnsportinfersets.gms
***     Output C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\gdxintrnsportinfersets.lst
***     PageWidth 80
***     PageSize 0
***     PageContr 3
***     Action CE
***     SubSys C:\GAMS\win64\24.1\gmscmpnt.txt
***     ScrDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\225a\
***     SysDir C:\GAMS\win64\24.1\
***     CurDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002\
***     WorkDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002\
***     PutDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002\
***     GridDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\225a\
***     ScriptNext 225a\gamsnext.cmd
***     ScriptFrst @echo off
***     LogOption 3
***     AppendOut 1
***     DFormat 0
***     TFormat 0
***     DumpParms 2
```

© 2022 Prof. Bruce McCarl

```
***     LibIncDir C:\GAMS\win64\24.1\inclib\
***     SysIncDir C:\GAMS\win64\24.1\
***     ErrorLog 99
***     IDE 1
***     ExecMode 1
```

This can be used in conjunction with the command line option `DumpParmsLogPrefix='string'` that will prefix the listing that are triggered by `DumpParms=2` with a specific string for easy filtering of these lines.

9.2.17 DumpParmsLogPrefix: DPLP

This keyword prefixes the list in the log file of all command line parameters changed or set by the user, GAMS or the IDE during the current run. String input is expected

The command is implemented with

```
DumpParmsLogPrefix=string
```

or

```
DPLP=string
```

This must be used in conjunction with the command line option `DumpParms=2` and is designed to allow one to easily find those lines. By default the lines are marked with ***.

For example using the command line parameters dp=2 DumpParmsLogPrefix=changed on the example model gdxintrnsportinfersets.gms results in a log file that contains the following

```
changed opennotype R=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.gms"
changed opentext    W=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.lxi"
changed opentext    W=0 FN="C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\gdxintrnsportinfersets.lst"
changed     LP CPLEX
changed     MIP CPLEX
changed     RMIP CPLEX
changed     MINLP DICOPT
changed     MIQCP DICOPT
changed     Input C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\gdxintrnsportinfersets.gms
changed     Output C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\gdxintrnsportinfersets.lst
changed     PageWidth 80
changed     PageSize 0
changed     PageContr 3
changed     Action CE
changed     SubSys C:\GAMS\win64\24.1\gmscmpnt.txt
changed     ScrDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\225a\
changed     SysDir C:\GAMS\win64\24.1\
changed     CurDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\
changed     WorkDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\
changed     PutDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs\gams2002
\
```

© 2022 Prof. Bruce McCarl

```
changed     GridDir C:\gamsstuff\mccarlguide\mccdoc\docs\bigdocs
\gams2002\225a\
changed     ScriptNext 225a\gamsnext.cmd
changed     ScriptFrst @echo off
changed     LogOption 3
changed     AppendOut 1
changed     DFormat 0
changed     TFormat 0
changed     DumpParms 2
changed     LibIncDir C:\GAMS\win64\24.1\incLib\
changed     SysIncDir C:\GAMS\win64\24.1\
changed     ErrorLog 99
changed     IDE 1
changed     ExecMode 1
changed     DumpParmsLogPrefix changed
```

9.2.18 EMP

This keyword controls the name of the solver that will be used to solve EMP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
EMP=solvername
```

9.2.19 Eolonly: Ey

This keyword controls formatting of parameters on the command line and is useful in conjunction with the pf parameter. Numeric input is expected with the allowable numeric values being

0	any number of keys or values (default)
1	only one key-value pair on a line

The command is implemented with

```
Eolonly=number
```

or

```
EY=number
```

9.2.20 Errmsg

This keyword controls the position of error messages in the listing file as discussed in the standard output and compilation error chapters. Numeric input is expected with the allowable numeric values being

0	Place error messages at the end of compiler listing (default)
1	Place error messages immediately following the line with the error
2	Suppress error messages

© 2022 Prof. Bruce McCarl

The command is implemented with the syntax

```
Errmsg=number
```

9.2.21 Errnam

This option specifies the name of a file defining the internally used compiler error messages. It is used to change the name from the default GAMSERRS.TXT. String input is expected.

The command is implemented with the syntax

```
Errnam=filename
```

This is not ordinarily changed by users.

9.2.22 Error

This keyword forces a parameter error with a specified message. String input is expected.

The syntax for this command is

```
Error=message
```

It is useful in conditional compilation exercises which contain error checking.

9.2.23 Errorlog: Er

This keyword controls the error message lines written to the LOG file. Numeric input is expected with the allowable numeric values being

`0`	no error messages to LOG file
n	Number of lines for each error that will be written to LOG file (default=99)

The command is implemented with the syntax

```
Errorlog=number
```

or

```
Er=number
```

9.2.24 Etlim: Etl

This GAMS parameter controls the time limit for a job. A GAMS job will terminate if the elapsed time in seconds exceeds the value of `Etlim`. Numeric input is expected

The command is implemented with the syntax

© 2022 Prof. Bruce McCarl

```
Etlim=number
```

or

```
Etl=number
```

The default value for Etlim is +inf.

The system will terminate with a compilation or execution error if the limit is reached.

9.2.25 Execerr

This keyword puts a maximum limit on the number of errors that can be found during execution or preprocessing associated with a solve statement. When more than Execerr errors have been found GAMS will abort. Numeric input is expected with the allowable numeric values being

0	no errors allowed limit (default)
n	max number allowed

The command is implemented with the syntax

```
Execerr=number
```

9.2.26 Execmode

This keyword limits users ability to run external programs or write to areas on the disk and is intended to help in network administration for security purposes. Numeric input is expected with the allowable numeric values being

0	everything allowed
1	interactive shells in $call and execute commands are prohibited
2	all $call and execute commands are prohibited
3	$echo/n or put commands can only write to directories in or below the working or scratch directory
4	$echo and put commands are not allowed

The command is implemented with the syntax

```
Execmode=number
```

9.2.27 Expand: Ef

The expand parameter generates a file that contains information about all the input files processed during a particular compilation. By default names are composed by completing the name with the working directory. String input is expected.

The command is implemented with the syntax

```
Expand=filename
```

© 2022 Prof. Bruce McCarl

or

```
Ef=filename
```

Example:

```
parameter a ; a = 0 ;
$include file2.inc
$include file2.inc
```

where file2.inc contains

```
a = a+1;
display a;
```

Running the model with the command line flag expand myfile.fil results in the creation of the myfile.fil with the following content

```
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 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.

9.2.28 FDDelta

This option allows one to exercise control over the computation of the hessian and numerical derivatives are computed. .

The default for `FDDelta` is 1e-5.. Numerical input is expected with the value needing to fall between 1.e-9 and 1.

The command is implemented with the syntax

```
Fddelta=value
```

© 2022 Prof. Bruce McCarl

Also users can control scaling of the step size using **FDOpt.**

9.2.29 FDOpt

This option allows one to exercise control over how numerical derivatives are computed in terns of scaling of steps, hessian calculation method, and use of numerical first derivatives**.** This involves a numerical parameter setting between 0 and 4 plus 10 id step size scaling is to be turned off . The default for `FDDelta` is 0.. Numerical input is expected with the value needing to fall between 0 -4 and 10-14.

The command is implemented with the syntax

```
Fdopt=value
```

The value involves specifying two digits: ij.

- The first i digit controls whether the step size (FDDelta) is scaled to reflect the size of the value of the point (x) around which the derivative is being formed. When i=0 the default state then scaling is used and d is set to the FDDelta value multiplied by max(1,|x|) (ie. d=max(1,|x|) *FDDelta). When i is set to 1 this turns off scaling and d=FDDelta.

- The j digit is mostly for testing, but allows one to influence gradient computation when the extrinsic function provides gradient but no Hessian values. By default the numerical derivatives routine uses the gradient calculation from the extrinsic function to approximate the Hessian. If the gradient is expensive to calculate compared to a function evaluation, it could be beneficial to use multiple function values to approximate the Hessian. In this case set the j digit to 1.

- Here are all possible values for FDopt:

  - 0: All derivatives are computed analytically if available, for numerical Hessian use gradient values, FDdelta will be scaled
  - 1: All derivatives are computed analytically if available, for numerical Hessian use function values, scale delta
  - 2: Gradient are computed analytically, force Hessian numerically using gradient values, scale delta (testing only)
  - 3: Gradient are computed analytically, force Hessian numerically using function values, scale delta (testing only)
  - 4: Force gradient and Hessian numerically, scale delta (testing only)
  - 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 (testing only)

© 2022 Prof. Bruce McCarl

- 13: Same as 3, but no scale of delta (testing only)
- 14: Same as 4, but no scale of delta (testing only)

Also users can control the step size using **FDDelta.**

9.2.30 Ferr

This option allows one to redirect compilation error messages to a file and names the file. The file name is composed by completing the name with the scratch directory and the scratch extension. Under default settings a file with compilation error messages is not generated. String input is expected.

The command is implemented with the syntax

```
Ferr=filename
```

9.2.31 Filecase

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 trnsport.REF if TRNSPORT.ref already exists. The command is implemented with the syntax

```
Filename=integer
```

where the allowable integer values are

0	causes GAMS to use default casing
1	causes GAMS to uppercase filenames
2	causes GAMS to lowercase filenames

9.2.32 FileStem

FileStem allows you to use a different stem name for LST, LXI, and LOG files along with any GDX files that use the default name.

Thus if you run myfile.gms but use the command line parameter

filestem=anothername

then the LST, LXI and LOG files will be named

anothername.lst, anothername.lxi, anothername.log

This allows users to save these files with a unique name when one is running multiple cases of single GAMS program in the same directory.

© 2022 Prof. Bruce McCarl

9.2.33 Forcework: 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. Numeric input is expected with the allowable numeric values being

```
0    no translation (default)
1    try translation
```

9.2.34 Forlim

This option 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. Numeric input is expected. The command is implemented with the syntax

```
Forlim=number;
```

as illustrated in **otheroptions.gms**. The default value is 2^32-1.

9.2.35 Fsave

This keyword forces GAMS to write a saved workfile. This allows one to save a file even in the face of execution or other errors. To work the SAVE option must be present and the file name is taken from the SAVE specification. GAMS will make up a name if the SAVE specification is not present. Numeric input is expected with the allowable numeric values being

```
0    workfile only written to file specified by SAVE if no errors occur (default)
1    workfile always written to file specified by SAVE or if SAVE is not present to a name made up by GA
```

The command is implemented with the syntax

```
Fsave=number
```

9.2.36 G205

This keyword controls the syntax allowed in GAMS causing reversions to older versions. 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. Numeric input is expected with the allowable numeric values being

```
0    use only latest syntax (default)
1    allow version 2.05 syntax only
2    allow version 2.25 syntax only
```

© 2022 Prof. Bruce McCarl

The command is implemented with the syntax

```
G205=number
```

9.2.37 Gdx

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. String input is expected.

The command is implemented with the syntax

```
Gdx=filename
```

Setting gdx to the string "default" i.e.

```
gdx=default
```

causes GAMS to create a GDX file with the GMS file root name and a GDX extension. Thus

```
gams trnsport gdx=default
```

will cause GAMS to write the GDX file trnsport.gdx.

9.2.38 Gdxcompress

This option allows the user to specify whether GDX files will be compressed or not. Numeric input is expected with the allowable numeric values being

0	do not compress gdx files (default)
1	compress gdx files

The command is implemented with the syntax

```
Gdxcompress=number
```

The command is entered on the command line of a DOS/UNIX GAMS call or in the parameters box in the IDE although an equivalent environmental variable exists. The command line parameter takes precedence.

Note that version 5 gdx files do not support compression.

9.2.39 Gdxconvert

This option allows the user to specify whether GDX files will be written in an older GDX file format . String input is expected with the allowable strings being

v5	version 5 gdx file, does not support compression
v6	version 6 gdx file
v7	version 7 gdx file (default)

© 2022 Prof. Bruce McCarl

The command is implemented with the syntax

```
Gdxconvert=v#
```

The command is entered on the command line of a DOS/UNIX GAMS call or in the parameters box in the IDE although an equivalent environmental variable exists. The command line parameter takes precedence.

9.2.40 GdxUELs

This option allows the user to specify whether only the UELs that are required by the exported symbols are registered to GDX or whether all will be registered. The command is entered on the command line of a DOS/UNIX GAMS call or in the parameters box in the IDE.

The command is implemented with the syntax

```
Gdxuels=squeezed
```

or

```
GdxUELs=full
```

`gdxUELs` is set to `squeezed` by default.

9.2.41 Griddir: Gdir

GDIR is a command line paramter that specifies the grid file directory. Each GAMS job has only one grid file directory. By default it is assumed to be the scratch directory. String input is expected.

The command is implemented with the syntax

```
Griddir=filename
```

or

```
Gdir=filename
```

If filename is not a fully qualified name, it will be completed using the current directory. This is also discussed in the grid computing section.

9.2.42 Gridscript: Gscript

A parameter that provides the name of a script file to submit grid computing jobs. String input is expected.

The command is implemented with the syntax

```
Gridscript=filename
```

© 2022 Prof. Bruce McCarl

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.

9.2.43 HeapLimit: Hl

Cases may arise where one needs to limit the amount of memory a GAMS job can use during compilation and execution. HeapLimit is a GAMS parameter with which one can limit the maximum amount of memory that can be used to store data during GAMS compilation and execution. If the needed data storage exceeds this limit, the job will be terminated with return code 10, out of memory.

The command is implemented with

```
Heaplimit=number
```

where number gives the memory use limit in megabytes. The default value is 1e20.

This can also be implemented through the function heaplimit.

The limit does not apply to solvers but CONOPT contains the option HeapLimit to allow memory limitations during the solve.

9.2.44 Holdfixed

This option can reduce the problem size by treating fixed variables as constants. Numeric input is expected with the allowable numeric values being

`0`	fixed variables are not treated as constants (default)
`1`	fixed variables are treated as constants

The command is implemented with the syntax

```
Holdfixed=number
```

9.2.45 Ide

The ide option instructs GAMS to write special instructions to the log file that are in turn read by the IDE. Numeric input is expected with the allowable numeric values being

© 2022 Prof. Bruce McCarl

1	causes the writing to occur (default when running under the IDE)
0	does not (default outside of IDE)

The command is implemented with

```
Ide=number
```

Normal IDE users do not need to worry about the ide parameter as GAMS adds it automatically.

It is of concern when Execute or $Call are used to spawn GAMS jobs as discussed here.

9.2.46 Input: 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. String input is expected.

The command is implemented with the syntax

```
Input=filename
```

or

```
I=filename
```

9.2.47 Inputdir: 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. String input is expected.

The command is implemented with

```
Inputdir=listofdirectorynames
```

or

```
Idir=listofdirectorynames
```

For example under Windows one could use

```
Idir=c:\;c:\gams;c:\mymodel
```

The search order is (1) current directory and (2) directories specified by inputdir in order.

© 2022 Prof. Bruce McCarl

9.2.48 Inputdir1 to inputdir40: Idir1 to 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. If not specified, the name will be blank. String input is expected.

The command is implemented with

```
Inputdir1=directoryname
Inputdir2=directoryname
```

or

```
Idir1= directoryname
Idir40= directoryname
```

For example under Windows one could use

```
Idir1=c:\  Idir2=c:\gams     Idir3=c:\mymodel
```

Where this corresponds to the example from the IDIR case above.

9.2.49 Integer1 to Integer5

Integer communication cell that can contain any integer number.

The command is implemented with

```
Integer1=integer
Integer2=integer
```

These parameters are only used in very special circumstances and generally not by most users.

The main usage to data has been for passing information to experimental solvers that don't have an option file that has been implemented inside the GAMS system.

9.2.50 IntVarUp

The default upper bound on integer variables has been changed from 100 to +INF but an option has been introduced for backward compatability. Namely since some previously developed GAMS applications may inadvertently depend on the previous default value of 100,, the GAMS command line parameter `IntVarUp=n`, has been introduced to allow the user to control what values will be passed to the solver. If the GAMS parameter `IntVarUp` is not specified then a default value of `IntVarUp=1` is assumed and the default bound will be 100 as in previous releases.

`IntVarUp=0`: A default upper bound of +INF will be passed to the solver for any integer variables without an explicit upper bound .

© 2022 Prof. Bruce McCarl

`IntVarUp`=1: A default upper bound of +100 will be passed to the solver for any integer variables without an explicit upper bound .. The solvers will operate as with older GAMS versions. In addition messages will be written to the log and listing to report on the number of integer or semi-integer variables which had the new default bound of +INF reset to 100.

`IntVarUp`=2: The new default values of +INF will be used as with `IntVarUp`=0. Additionally when a solution is returned to GAMS and the level value of an integer variable exceeds the old bound value of 100, a message will be written to the log and listing files.

`IntVarUp`=3: The same as `IntVarUp`=2 with an additional execution error issued if the solution reports a level value greater than 100 for any integer variable with a default bound of +INF.

Implementation - to use a default upper bound of +INF you can use the following command line parameter:

`IntVarUp=0`

Setting `IntVarUp` values to 2 and 3 is a convenient way to test if the application run under older GAMS versions would have been constrained by the older default bound of of 100. Future releases may use `IntVarUp`=0 as the default. Note that, one can use a shorted tect string for this of `PF4 i.e. pf4=2`

9.2.51 iterlim

This parameter specifies the maximum number of allowable solver iterations, before the solver terminates the run. Numeric input is expected.

The command is implemented with

```
Iterlim=number
```

The default value is 2000000000.

9.2.52 Jobtrace: Jt

This parameter specifies a string written to the trace file at the end of a GAMS job. String input is expected.

The command is implemented with

```
Jobtrace=string
```

© 2022 Prof. Bruce McCarl

or

```
Jt=string
```

9.2.53 Keep

This keyword tells GAMS whether to keep the temporary files generated during a run. Numeric input is expected with the allowable numeric values being

0	delete all files (default)
1	keep all temporary files

The command is implemented with the syntax

```
Keep=number
```

9.2.54 Libincdir: Ldir

This keyword gives the 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 not specified, it will be set to the inclib subdirectory of the GAMS system directory. A relative path is relative to the GAMS System Directory. String input is expected.

The command is implemented with

```
Libincdir=directoryname
```

or

```
Ldir= directoryname
```

Notes:

- Unlike ldir, additional directories cannot be set with Ldir. The string passed will be treated as one directory. Passing additional directories will cause errors.
- If the Ldir parameter is set, the default library include directory is not searched.

9.2.55 License

This option specifies the name of file containing the GAMS license. It is used to change the name from the default GAMSLICE.TXT located in the system directory. String input is expected.

The command is implemented with

```
License=Licensefilename
```

9.2.56 Limcol

This keyword controls the number of cases written to the LST file for each named variable in a model.

© 2022 Prof. Bruce McCarl

The command is implemented with

```
Limcol=number
```

The default value is 3.

9.2.57 Limrow

This keyword controls the number of cases written to the LST file for each named equation in a model.

The command is implemented with

```
Limrow=number
```

The default value is 3.

9.2.58 Logfile: Lf

This option specifies the name of the LOG file. The LOG file name is completed using the working directory. If no Logfile is given but the LogOption=2 or LogOption=4, then the file name will be the input file name with the extension LOG. String input is expected.

The command is implemented with the syntax

```
Logfile=filename
```

or

```
Lf= filename
```

9.2.59 Logline: Ll

This keyword controls the amount of line tracing to the LOG file. Numeric input is expected with the allowable numeric values being

0	all line tracing suppressed
1	shows include file nesting
2	full line tracing (default)

The command is implemented with the syntax

```
Logline=number
```

or

```
LL=number
```

© 2022 Prof. Bruce McCarl

9.2.60 Logoption: Lo

This keyword tells GAMS whether to write a LOG file to the screen. Numeric input is expected with the allowable numeric values being

0	suppress LOG output
1	LOG output to screen (default)
2	send LOG output to file
3	writes LOG output to standard output
4	writes LOG output to a file and standard output

This can be used with a setting of 0 or 2 to permit jobs to operate in background.

9.2.61 LP

This keyword controls the name of the solver that will be used to solve LP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
LP=solvername
```

9.2.62 MaxProcDir

This keyword gives the maximum number of workfile directories that can be generated by GAMS. By default they are called 225a, 225b, ..., 225aa, 225ab .... Numeric input is expected.

The command is implemented with

```
MaxProcdir=number
```

where number is an integer.

For example MaxProcDir=100 will allow 100.

The 225 lablel can be changed using the parameter procdir.

9.2.63 MCP

This keyword controls the name of the solver that will be used to solve MCP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
MCP=solvername
```

© 2022 Prof. Bruce McCarl

9.2.64 McprHoldFx

And attribute, option or command line parameter which controls whether when dealing with an MCP problem a list of equations is generated and output that is designed to help resolving with complementary matching issues. Such issues can arise whensome variables are fixed in value using the .FX command and the holdfixed parameter is used. For more discussion see here.

The command is implemented with the syntax

```
MCPrHoldFX=1
```

9.2.65 MINLP

This keyword controls the name of the solver that will be used to solve MINLP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
MINLP=solvername
```

9.2.66 MIP

This keyword controls the name of the solver that will be used to solve MIP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
MIP=solvername
```

9.2.67 MIQCP

This keyword controls the name of the solver that will be used to solve MIQCP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
MIQCPP=solvername
```

9.2.68 MPEC

This keyword controls the name of the solver that will be used to solve MPEC models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
MPEC=solvername
```

© 2022 Prof. Bruce McCarl

9.2.69 Multipass: Mp

This is a command line parameter keyword that tells GAMS whether to use quick syntax 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. Numeric input is expected with the allowable numeric values being

```
0       employ standard compilation
1       quick check-out compilation
2       does the quick check-out compilation but skips all incidences of $call and ignores any missing files
```

The command is implemented with the syntax

```
Mulitpass=number
```

or

```
MP=number
```

9.2.70 NLP

This keyword controls the name of the solver that will be used to solve NLP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
NLP=solvername
```

9.2.71 Nodlim

This parameter specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. Numeric input is expected.

The command is implemented with

```
Nodlim=number
```

The default value is 0 and interpreted as 'not set'.

9.2.72 NoNewVarEqu

This parameter specifies that GAMS is to run in a mode that does not permit new variables or equations to be defined.. Numeric input is expected.

The command is implemented with

```
Nonewvarequ=0
or
Nonewvarequ=1
```

© 2022 Prof. Bruce McCarl

The default value is 0 and interpreted as variable and equation is allowed. Setting will cause GAMS to trigger compilation errors if the user tries to define new equation or variable symbols.

This is useful for testing GAMS run-time environments.

9.2.73 On115

Generate errors for unknown unique element in an equation. Numeric input is expected with the allowable numeric values being

`0`	no error messages (default)
`1`	issue error messages

The command is implemented with

```
On115=number
```

9.2.74 Optca

Absolute optimality criterion. This attribute specifies an absolute termination tolerance for use in solving all mixed-integer models. Numeric input is expected.

The command is implemented with

```
Optca=realnumber
```

The default value is 0.0.

9.2.75 Optcr

Relative optimality criterion. This attribute specifies a relative termination tolerance for use in solving all mixed-integer models. Numeric input is expected.

The command is implemented with

```
Optcr=realnumber
```

The default value is 0.1.

9.2.76 Optdir

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. String input is expected.

The command is implemented with

```
Optdir=directoryname
```

© 2022 Prof. Bruce McCarl

9.2.77 Optfile

This option specifies the default number for modelname.optfile in the code. Integer input is expected.

The command is implemented with the syntax

```
Optfile=number
```

9.2.78 Output: O

This option specifies the name of file containing the output. If no name is given, the input file name is combined with the working directory and the standard output file extension (LST) is applied. Otherwise, the final name is composed by using the working directory. String input is expected.

The command is implemented with the syntax

```
Output=filename
```

or

```
O=filename
```

9.2.79 Pagecontr: Pc

This keyword affects the page control in the listing file. Numeric input is expected with the allowable numeric values being

0	no page control with padding
1	Fortran style line printer format
2	no page control, no padding
3	Form feed character for new page (default)

It is implemented using the syntax

```
Pagecontr=number
```

or

```
Pc=number
```

9.2.80 Pagesize: Ps

This keyword tells the number of lines per page for printing the listing file. The default value is 58 and the lower bound is 0 which is interpreted as '+inf'.

It is implemented using the syntax

```
Pagesize=number
```

or

```
Ps=number
```

© 2022 Prof. Bruce McCarl

9.2.81 Pagewidth: Pw

This option sets the print width on a page in the listing file with a possible range from 72 to 32767 and a default value of 255. 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.

It is implemented using the syntax

```
Pagewidth=number
```

or

```
Pw=number
```

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.

9.2.82 Parmfile: 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.

The command is implemented with the syntax

```
Parmfile=filename
```

or

```
Pf=filename
```

It is read from the current directory unless a path is specified.

One thus could use

```
Gams myfile pf=c:\place\fileofcommands
```

Where file of commands was one like

```
Eolonly 1
Ps=9999
Pw=90
Errmsg=1
Pf=c:\place\moreparameters
Eolonly 0
```

Use of the eolonly parameter allows one to place one command per line.

This can be used in a customization exercise with the same file used by default in the IDE or in a

© 2022 Prof. Bruce McCarl

gmsprmXX.txt file as discussed in the chapter on customization. As shown in blue above this parameter may be used recursively with the pf file containing a pf= line identifying a file containing further pf entries.

9.2.83 Plicense

This keyword tells the name of a privacy license file that contains file encryption codes. It is implemented using the syntax

```
plicense=targetfilename
```

A full path should be used.

One thus could use

```
Gams myfile plicense=c:\place\speciallicensefile.txt
```

Where speciallicensefile.txt is obtained from GAMS corporation as discussed in the **User's Guide** or the compression and encryption section.

9.2.84 ProcDir

This keyword gives the name of the directory where the work files generated by GAMS will be stored. This directory must already exist, otherwise an error is triggered. If not specified, it will be set to 225 with characters a,b,c,... attached. String input is expected.

The command is implemented with

```
Procdir=name
```

The internal item %gams.procdir% is also defined and gives the name of the actual process directory in use for this run.

9.2.85 Profile

This keyword tells GAMS whether to include statement execution time and memory use reporting to the LST file. Numeric input is expected with the allowable numeric values being

0	no profiling (default)
1	minimum profiling
2	profling 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.

The command is implemented with the syntax

© 2022 Prof. Bruce McCarl

```
Profile=number
```

At the end of the log file a profile summary is created which contains (up to) ten of the 'worst' control structures.

Examples:

Running [TRNSPORT] 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.

9.2.86 Profilefile

A parameter that causes profiling information to be written to a file.

The expected input is

```
PROFILEFILE=file
```

where `file` is a file name

This allows one to write profiling information to a text file with a fixed format which can easily be imported into a spreadsheet for further analysis.

Note that profiling information is only created with `profile=1` or `profile=2`.

9.2.87 Profiletol: Ptol

This option specifies the minimum time a statement must use to appear in the profile generated output. The default value is 0.0.

It is implemented using the syntax

```
Profiletol=number
```

© 2022 Prof. Bruce McCarl

or

```
Ptol=number
```

9.2.88 Putdir: Pdir

This keyword gives the name of the directory where files generated by the Put command will be stored. If not specified, it will be set to the current working directory. String input is expected.

The command is implemented with

```
Putdir=directoryname
```

or

```
Pdir=directoryname
```

9.2.89 QCP

This keyword controls the name of the solver that will be used to solve QCP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
QCP=solvername
```

9.2.90 Reference: Rf

This option specifies the name of the file on which to place extensive reference map information. New information is appended to this file. String input is expected.

The command is implemented with the syntax

```
Reference=filename
```

or

```
Rf=filename
```

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 rf=default` will cause GAMS to write the reference file trnsport.ref.

9.2.91 Reslim

This parameter specifies the maximum time in seconds that the computer can run during execution of a solver, before the solver terminates the run. Numeric input is expected.

The command is implemented with

```
Reslim=number
```

© 2022 Prof. Bruce McCarl

The default value is 1000.

9.2.92 Restart: R

This option specifies the name of file containing a workfile written by a SAVE command that allows the GAMS program to be restarted. String input is expected.

The command is implemented with the syntax

```
Restart=filename
```

or

```
R=filename
```

For example one can use

```
r=myfile
```

where GAMS will create myfile.go0

For more detailed information see the Save and Restart basics.

9.2.93 RMINLP

This keyword controls the name of the solver that will be used to solve RMINLP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
RMINLP=solvername
```

9.2.94 RMIP

This keyword controls the name of the solver that will be used to solve RMIP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
RMIP=solvername
```

9.2.95 RMIQCP

This keyword controls the name of the solver that will be used to solve RMIQCP models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

© 2022 Prof. Bruce McCarl

```
RMIQCP=solvername
```

9.2.96 RMPEC

This keyword controls the name of the solver that will be used to solve RMPEC models. String input is expected with the solver name being one of the known GAMS solvers and that solver being capable of solving this type of model.

The command is implemented with

```
RMPEC=solvername
```

9.2.97 Save: 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. String input is expected.

The command is implemented with the syntax

```
Save=filename
```

or

```
S=filename
```

For more detailed information see the Save and Restart basics.

9.2.98 Savepoint: Sp

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. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point. Numeric input is expected with the allowable numeric values being

0	no point gdx file is to be saved (default)
1	a point gdx file is to be saved from the last solve in the GAMS model
2	a point gdx file is to be saved from every solve in the GAMS model

The command is implemented with the syntax

```
Savepoint=number
```

or

```
SP=number
```

When Savepoint=1 the point gdx file saved has the name modelname_p.gdx so for a model identified in the solve statement as transport the file would be transport_p.gdx. On the other hand if Savepoint=2 then the file name is modelname_pnn.gdx where nn is the solve number as determined internally by GAMS (**gdxsavepoint2.gms**). Thus, for a model solved 2 times that is identified with the name firm in the solve statement, the file names would be firm_p1.gdx and firm_p2.gdx. The file is reloaded with the Execute_loadpoint syntax.

© 2022 Prof. Bruce McCarl

This can also be done through an option command or a model attribute.

9.2.99 Scrdir: Sd

This keyword gives the name of the directory to be used by GAMS for temporary files generated during execution. If not specified, it will be set to a subdirectory of the current directory with an internally generated name. String input is expected.

The command is implemented with

```
Scrdir=directoryname
```

or

```
Sd= directoryname
```

9.2.100 Screxit: Se

This parameter gives the name of the extension for the GAMS temporary files generated during execution. String input is expected.

The command is implemented with

```
Screxit=string
```

or

```
Se=string
```

The default is `Screxit=dat`.

9.2.101 Scriptexit

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)).

String input is expected.

The command is implemented with

```
Scriptexit=string
```

9.2.102 Scriptfrst: Sf

This parameter specifies the first line written to gamsnext. The default is an empty string and the first line is not written.

String input is expected.

© 2022 Prof. Bruce McCarl

The command is implemented with

```
Scriptfrst=string
```

or

```
Sf=string
```

9.2.103 Scriptnext: Script

Gams Parameter that specifies the script mailbox file name. String input is expected.

The command is implemented with

```
Scriptnext=string
```

or

```
Script=string
```

The default is `script=gamsnext`.

9.2.104 Scrnam: 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. String input is expected.

The command is implemented with

```
Scrnam=string
```

or

```
Sn=string
```

9.2.105 Seed

This option specifies the seed used for the pseudo random number generator. Numeric input is expected.

The command is implemented with

```
Seed=number
```

The default value is 3141.

9.2.106 Solprint

This keyword controls the printing of the solution report. Numeric input is expected with the allowable numeric values being

`0` remove solution listings following solves

© 2022 Prof. Bruce McCarl

1 include solution listings following solves (default)
2 suppress all solution information

The command is implemented with the syntax

```
Solprint=number
```

A related set of Solprint constants is defined as follows

`Solprint.Summary` is a constant that equals 0.

`Solprint.Report` is a constant that equals 1.

`Solprint.Quiet` is a constant that equals 2.

These may be used in a fashion as follows (simpnlp.gms)

```
a1= %Solprint.Summary%;
a2=  %Solprint.Report%;
a3=  %Solprint.Quiet%;
a4=  %gams.Solprint%;
display a1,a2,a3,a4;
if(%gams.solprint%= %Solprint.Report%,display 'Normal output present');
display 'gams.solprint =','%gams.solprint%';
```

These constants are defined at compile time and cannot be manipulated or reassigned.

9.2.107 Solvelink: Sl

This option controls GAMS function when linking to solve. Numeric input is expected with the allowable numeric values being

0 GAMS operates as it has for years (default)
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 wait (same submission process as 3) in a Grid computing environment. This is included mainly for debugging purposes as is solvelink=7.
5 the problem is passed to the solver in core without use of temporary files
6 GAMS uses multi-threading and does not wait for the solver to return the solution and does not automatically collect the result when a solve statement is executed. Instead, the model is generated and then passed to the solver in a separate thread while GAMS continues the execution. This way, multiple models can be solved in parallel and the results can be collected later. This is similar to what is done under the Grid Computing

© 2022 Prof. Bruce McCarl

Facility with the difference, that the solver does not operate in its own process space but in a separate thread. After the solve statement, one can store a handle of the model instance using the model attribute mymodel.handle and use the functions that are used for the Grid Facility to collect the solution. One can use the command line parameter ThreadsAsync to reserve threads for processes. Only selected solvers can be used as listed below

7 GAMS submits a single job to the solver using the same linking procedure as Solvelink=6 but waits for the solution to be returned. This is included mainly for debugging purposes as is solvelink=4.

Currently, the following solvers can be used with SolveLink = 6:

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

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

The command is implemented with

Solvelink=number

Leaving GAMS open or passing the information in core saves time. On the other hand additional memory is required. This option is best for jobs that have a large data set and solve many small models as in that case one sacrifices memory but avoids the overhead of many GAMS saves and restarts. This is implemented by using the option SOLVELINK that can appear on the command line, as a model attribute or as an internal option statement.

The default setting is zero.

A set of SolveLink constants have been defined

Solvelink.ChainScript is a constant that equals 0.
Solvelink.CallScript is a constant that equals 1.
Solvelink.CallModule is a constant that equals 2.
Solvelink.AsyncGrid is a constant that equals 3.
Solvelink.AsyncSimulate is a constant that equals 4.
Solvelink.LoadLibrary is a constant that equals 5.

These may be used as follows

modelname.solvelink=%solvelink.AsyncGrid%;

which is the same as setting the item to 3

© 2022 Prof. Bruce McCarl

These constants are defined at compile time and cannot be manipulated or reassigned.

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

This can also be done through a model attribute or an option command.

9.2.108 Solver

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=bdmlp` 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=bdmlp` behaves identically to `solver=bdmlp lp=conopt`). If multiple occurrences of option solver appear, the last one sets the option as it is with other options, including LP, MIP, ...

9.2.109 Solvercntr: Scntr

This parameter specifies the solver control file name. String input is expected.

The command is implemented with

```
Solvercntr=text
```

or

```
Scntr=text
```

9.2.110 Solverdict: Sdict

This parameter specifies the solver dictionary file name. String input is expected.

The command is implemented with

```
Solverdict=text
```

or

```
Sdict=text
```

9.2.111 Solverinst: Sinst

This parameter specifies the solver instruction file name. String input is expected.

The command is implemented with

```
Solverinst=text
```

or

© 2022 Prof. Bruce McCarl

```
Sinst=text
```

9.2.112 Solvermatr: Smatr

This parameter specifies the solver matrix file name. String input is expected.

The command is implemented with

```
Solvermatr=text
```

or

```
Smatr=text
```

9.2.113 Solversolu: Ssolu

This parameter specifies the solver solution file name. String input is expected.

The command is implemented with

```
Solversolu=text
```

or

```
Ssolu=text
```

9.2.114 Solverstat: Sstat

This parameter specifies the solver status file name. String input is expected.

The command is implemented with

```
Solverstat=text
```

or

```
Sstat=text
```

9.2.115 Stepsum

This parameter controls the generation of a step summary of the processing times taken by GAMS during a given run. Numeric input is expected with the allowable numeric values being

`0`	no step summary (default)
`1`	step summary printed

The command is implemented with the syntax

```
Stepsum=number
```

Example:

To illustrate the use of the stepsum option, the default GAMS LST file from running [TRNSPORT] with the option

© 2022 Prof. Bruce McCarl

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.

9.2.116 strictSingleton

This option specifies whether GAMS is to trigger out an error if more than one element is specified in a singleton set. Integer input is expected.

The command is implemented with the syntax

```
Strictsingleton=number
```

Where the values and corresponding actions for the number entry are.

0	Use the first set element ignoring the rest when the singleton set has multiple elements
1	Message out an error if assignment to singleton set causes the set to have multiple elements

The default value is one

9.2.117 Stringchk

This keyword tells GAMS how to perform a string substitution check for %xxx% symbols. Numeric input is expected with the allowable numeric values being

0	no substitution if symbol undefined and no error (default)
1	error if symbol undefined
2	remove %xxx% if symbol undefined and no error

The command is implemented with the syntax

© 2022 Prof. Bruce McCarl

```
Stringchk=number
```

9.2.118 Subsys

This option specifies the configuration file name that contains solver defaults and other information. It is used to change the name of GMSCMPNT.TXT on Windows or GMSCMPUN.TXT on Unix. The name will be used as is. String input is expected.

The command is implemented with the syntax

```
Subsys=filename
```

9.2.119 Suppress

This keyword tells GAMS whether to suppress the compiler echo print of the GAMS input instructions. This parameter is similar in functionality to the $offlisting dollar control option. Numeric input is expected with the allowable numeric values being

0	standard compiler listing (default)
1	suppress compiler listing

The command is implemented with the syntax

```
Suppress=number
```

9.2.120 Symprefix

This option specifies a prefixes that is appended to the beginning of all user symbols that are in this run in a resultant save/restart file. This avoids name clashes when merging multiple models together. String input is expected.

The command is implemented with the syntax

```
Symprefix=filename
```

9.2.121 Symbol

This option specifies the name of a partial symbol table that can be written in conjunction with reference files. String input is expected.

The command is implemented with the syntax

```
Symbol=filename
```

9.2.122 Sys10

This option controls whether GAMS converts exponentiation treating a real power as an integer power if the exponent is constant and within 10^{-12} of an integer value. This option is used by setting

© 2022 Prof. Bruce McCarl

```
Sys10=number;
```

where two numerical values are allowed

0	which does not convert exponentiation.
1	which converts exponentiation.

The default value is 0.

there is a corresponding option command implementd by

option sys10=number;

9.2.123 Sys11

Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use. Numeric input is expected with the allowable numeric values being

0	automatic optimization/restructuring of data
1	no optimization
2	always optimize/restructure

The command is implemented with the syntax

```
Sys11=number
```

9.2.124 Sysdir

This keyword gives the name of the directory where GAMS executables reside. If not specified, it will be set to the GAMS system directory. String input is expected.

The command is implemented with

```
Sysdir=directoryname
```

9.2.125 Sysincdir: Sdir

This keyword gives the name of the directory to be used by GAMS for sysinclude files that do not have a full path specification. If not specified, it will be set to the GAMS system directory. String input is expected.

The command is implemented with

```
Sysincdir=directoryname
```

or

© 2022 Prof. Bruce McCarl

```
Sdir= directoryname
```

9.2.126 Sysout

This option controls the incorporation of additional solver generated output (that in the solver status file) to the listing file. Numeric input is expected with the allowable numeric values being

0	suppress additional solver generated output (default)
1	include additional solver generated output

The command is implemented with the syntax

```
Sysout=number
```

9.2.127 Tabin

This keyword tells GAMS how to deal with tabs. Numeric input is expected with the allowable numeric values being

0	tabs are not allowed
1	tabs are replaced by blanks
n	tabs are 1, n+1, 2n+1,.. (default: n=8)

The command is implemented with the syntax

```
Tabin=number
```

9.2.128 Tformat: Tf

This keyword controls compiler use of alternative time formats. Numeric input is expected with the allowable numeric values being

0	hh:mm:ss (default)
1	hh.mm.sd

The command is implemented with

```
Tformat=number
```

or

```
Tf=number
```

9.2.129 Threads

Threads is an option that controls the number of threads or CPU cores to be used by a solver. It can be set as a command line parameter, a GAMS option or a model attribute. Note that this does not alter the cores used in Grid computing. Numeric input is expected with the allowable numeric values being

-n	reserve n cores

© 2022 Prof. Bruce McCarl

```
0       use all available cores
n       use n cores
```

it is implemented in the command line with the syntax

```
Threads=number
```

or as an option with

```
option threads=number;
```

or as a model attribute with

```
modelname.threads=number;
```

Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

9.2.130 ThreadsAsync

ThreadsAsync is an command line parameter that controls the number of threads or CPU cores to use (at most) for asynchronous solve (SolveLink=6) .

The Default value is: -1.

The following values are allowed

value	meaning
0	use all available cores
n	use n cores
-n	number of cores to leave free for other tasks

Further documentation on the use of this feature appears in the Grid computing section and in the document here.

© 2022 Prof. Bruce McCarl

9.2.131 Timer

This option specifies an instruction timer threshold in milli seconds. That means that only details about internal GAMS intructions that took more than n milli seconds are echoed to the log.

The command is implemented with

```
Timer=number
```

The default 0.0 is interpreted as +inf and no details are echoed.

9.2.132 Trace

This option specifies the trace file name and causes a trace file to be written. String input is expected.

The command is implemented with the syntax

```
Trace=filename
```

9.2.133 Traceopt

This option specifies the trace file format. Numeric input is expected with the allowable numeric values being

0	solver and GAMS step trace without headers
1	solver and GAMS step trace
2	solver step trace only
3	trace file format used for GAMS performance world
5	trace file with all available trace fields

The command is implemented with the syntax

```
Traceopt=number
```

9.2.134 User1 to user5: U1 to U5

This keyword permits user entry of text for up to 5 user-defined options. String input is expected.

The command is implemented with the syntax

```
User1=string
User2=string
```

© 2022 Prof. Bruce McCarl

or

```
U1=string
U5=string
```

9.2.135 Warnings

This option specifies the maximum number of allowable warnings, before the run terminates. Numeric input is expected. The default value is MAXINT.

9.2.136 Workdir: Wdir

This keyword gives the name of the working directory to be used by GAMS. If not specified, it will be set to the Curdir name. String input is expected.

The command is implemented with

```
Workdir=directoryname
```

or

```
Wdir= directoryname
```

9.2.137 Workfactor

This keyword gives the multiplier to initially allocate for solver memory as a multiple of the GAMS memory use estimate. The default value is 1.0. Real input is expected.

The command is implemented with

```
Workfactor=realnumber
```

If one sets workfactor to 2.5 then 2.5 times the GAMS memory use estimate for a problem will be allocated.

9.2.138 Workspace

Parameter that when initializes all modelname.workspace attributes.

The command is implemented with

```
Workspace=realnumber
```

where realnumber is memory space in megabytes.

If not given by the user the solver can choose the size and usually it is the memory available on the machine.

© 2022 Prof. Bruce McCarl

9.2.139 Xsave: Xs

This option specifies the name of a save file written in ASCII format in older systems (versions older than 21.7), so it is platform independent and can be moved to machines with different operating systems. String input is expected.

In GAMS systems from release 22.3 and newer it causes writing of compressed save files.

The command is implemented with the syntax

```
Xsave=filename
```

or

```
Xs=filename
```

9.2.140 Zerores

This parameter specifies the threshold value for internal rounding to zero in certain operations. Numeric input is expected.

The command is implemented with

```
Zerores=number
```

The default value is 0.0.

9.2.141 Zeroresrep

This option causes GAMS to issue warnings whenever a rounding occurs because of zerores. Numeric input is expected with the allowable numeric values being

0	no warning when a rounding occurs because of zerores (default)
1	issue warnings whenever a rounding occurs because of zerores

The command is implemented with the syntax

```
Zeroresrep=number
```

10 Saves and Restarts

In GAMS one may divide a job into pieces and reassemble them as if they were all continuous using the GAMS save and restart command line parameters.

© 2022 Prof. Bruce McCarl

Basics
Use of save and restarts and their effect
Why use save and restart?

10.1 Save Restart Basics

Suppose we have a program like **transml.gms** and divide it into three parts **trandata.gms** the red part below, tranmodl.gms the blue part below and **tranrept.gms** the magenta part below.

```
SETS  PLANT    PLANT LOCATIONS  /NEWYORK  , CHICAGO , LOSANGLS /
      MARKET   DEMAND MARKETS   /MIAMI,   HOUSTON, MINEPLIS, PORTLAND/
PARAMETERS   SUPPLY(PLANT)  QUANTITY AVAILABLE AT EACH PLANT
               /NEWYORK   100, CHICAGO   275, LOSANGLS   90/
             DEMAND(MARKET)   QUANTITY REQUIRED BY DEMAND MARKET
               /MIAMI 100,HOUSTON 90,MINEPLIS 120,PORTLAND 90/;
TABLE   DISTANCE(PLANT,MARKET)   DISTANCE FROM EACH PLANT TO EACH MARKET
                     MIAMI   HOUSTON   MINEPLIS   PORTLAND
         NEWYORK      1300     1800       1100       3600
         CHICAGO      2200     1300        700       2900
         LOSANGLS     3700     2400       2500       1100      ;
PARAMETER COST(PLANT,MARKET)    CALCULATED COST OF MOVING GOODS;
          COST(PLANT,MARKET) = 50 + 1 * DISTANCE(PLANT,MARKET);
POSITIVE VARIABLES
         SHIPMENTS(PLANT,MARKET) AMOUNT SHIPPED OVER A TRANSPORT ROUTE;
VARIABLES TCOST                  TOTAL COST OF SHIPPING OVER ALL ROUTES;
EQUATIONS TCOSTEQ                TOTAL COST ACCOUNTING EQUATION
         SUPPLYEQ(PLANT)         LIMIT ON SUPPLY AVAILABLE AT A PLANT
         DEMANDEQ(MARKET)        MINIMUM REQUIREMENT AT A DEMAND MARKET;
TCOSTEQ.. TCOST =E=SUM((PLANT,MARKET), SHIPMENTS(PLANT,MARKET)*
                                          COST(PLANT,MARKET));
SUPPLYEQ(PLANT).. SUM(MARKET,SHIPMENTS(PLANT,MARKET))=L=SUPPLY(PLANT);
DEMANDEQ(MARKET)..SUM(PLANT,SHIPMENTS(PLANT,MARKET))=G=DEMAND(MARKET);
MODEL TRANSPORT /ALL/;
SOLVE TRANSPORT USING LP MINIMIZING TCOST;
ParAMETER MOVEMENT(*,*)  COMMODITY MOVEMENT;
MOVEMENT(PLANT,MARKET)=SHIPMENTS.L(PLANT,MARKET);
MOVEMENT("TOTAL",MARKET)=SUM(PLANT,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT(PLANT,"TOTAL")=SUM(MARKET,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT("TOTAL","TOTAL")=SUM(MARKET,MOVEMENT("TOTAL",MARKET));
OPTION DECIMALS=0;
DISPLAY MOVEMENT;
```

We could execute this using an include file sequence such as in **tranint.gms**. We can also use save and restart files. First let me introduce the needed commands.

Save: S

© 2022 Prof. Bruce McCarl

Restart: R
Xsave: Xs

10.1.1 Save: S

Save is a GAMS command line parameter. It is used though the syntax

```
Save=path\filestemname
```

or

```
S=path\filestemname
```

Where path may be used or omitted and filestemname is the name under which a family of seven to nine work files will be saved. In versions before 20.1 these files were named

```
Path\filestemname.G01
Path\filestemname.G02
…
Path\filestemname.G09
```

and will be platform dependent binary files only suitable for GAMS use but not for user manipulation. In later versions there is only one of these and it in a packed form that is machine independent. That file is named

```
Path\filestemname.G00
```

If the path is omitted the files will be saved in the current working directory. Any valid name for the filestemname may be employed which will result in a validly named file in the operating system being used.

Often it is useful to include a path to a directory where these will remain out of the way. Such a path is achieved often by using .\t\ which places the work files in a subdirectory of the current working directory named t.

The work files contain the complete contents of the GAMS run after execution and also may contain items from earlier GAMS runs if incorporated through a restart command.

Current work files are platform independent and may be moved for example from a windows machine to a Unix or Linux machine. For versions of GAMS before 21.7 Xsave as discussed below allows platform independence.

10.1.2 Restart: R

Restart is a GAMS command line parameter. It is used though the syntax

```
Restart=path\filestemname
```

or

```
R=path\filestemname
```

© 2022 Prof. Bruce McCarl

where the path and filestemname must match the characteristics of a set of work files that have already been saved through a save command.

- Restart senses whether the file it starts up from is a Save type or an Xsave type and thus can read both.

10.1.3 Xsave: Xs

Xsave is an older pre version 21.7 GAMS command line parameter which provides an alternative to Save. It is used though the syntax

```
Xsave=path\filestemname
```

or

```
Xs=path\filestemname
```

where path may be used or omitted and filestemname is the name under which a family of up to nine work files will be saved. These files follow all the naming characteristics of the save files discussed above. In terms of composition they contain the same information being still only suitable for internal GAMS use not for user manipulation, but were platform independent but Save now generates such files.

Notes:

- Wen using restart GAMS automatically senses whether the file it starts up from is a Save type or an Xsave type and thus can read both.

10.2 Use of save and restarts and their effect

Saves and restarts allow one to run GAMS code up to a point and save the job status in a set of work files, then start another job which has all the results of the first job. Thus, for example, we could divide a big job into pieces running the data component (**trandata.gms**) then using save to preserve the run status in work files called S1. Then we could run the model component **tranmodl.gms** restarting from the S1 work files and save the resultant status in work files called S2. Finally we could run the report writing part **tranrept.gms** restarting from the work files in S2. The net effect of this is the same as if the continuous undivided file was run composed of the three parts in the individual files. Each component would start from the previous component just if it was one continuous file.

Save and restart on command line
IDE usage

10.2.1 Save and restart on command line

In DOS and Unix commands to invoke save and restart look like

```
GAMS trandata              s=s1
GAMS tranmodl    r=s1      s=s2
GAMS tranrept    r=s2
```

© 2022 Prof. Bruce McCarl

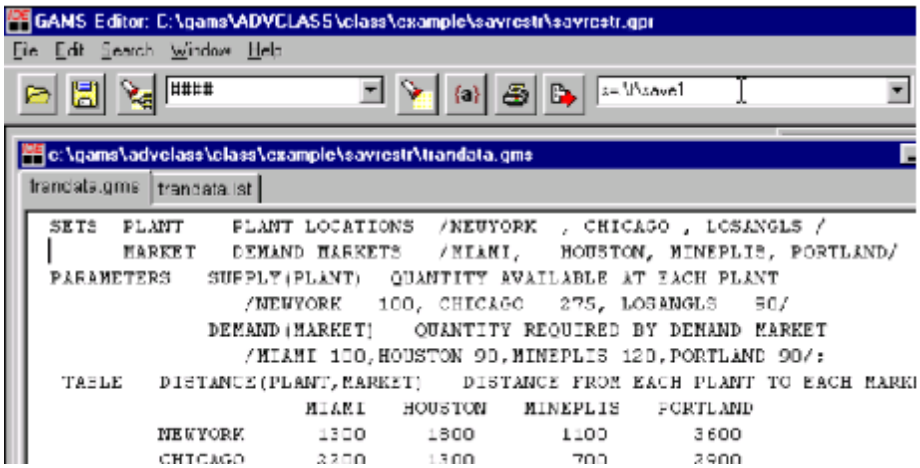
These are implemented in the file **tran.bat** with one small addition. Namely, the save and restart are given a relative address (e.g. .\t\s1) so they're put in a subdirectory t under the area where we are working to avoid clutter. When GAMS is told to save its files it generates 7-9 files for each saved set of work files.

One can also use xsave to save machine independent files

```
GAMS trandata              xsave=s1
GAMS tranmodl    r=s1      xsave=s2
GAMS tranrept    r=s2
```

10.2.2 IDE usage

Saves and restarts can also be used in the IDE in the GAMS command line parameters box. Entries in that box associate a set of execution time command line parameters with a file that are automatically added to the GAMS call.



The IDE saves any entries in the command line parameter box in the current IDE project. Thus, once you specify the save and restart parameters you will have these associated with every subsequent use of the file provided you're using the right project and have not changed the restart information.

10.2.2.1 Save and restart calling GAMS from within GAMS

One may not like the IDE command line option since it requires one to wait for one job before beginning the next. One can overcome this by using GAMS to call itself. To do this we enter GAMS command line instructions within quotes in an Execute statement as follows (**saverestar.gms**) where I am also using the $log command to send a message to the IDE process window.

```
execute "GAMS trandata              s=s1"
$log done1
execute "GAMS tranmodl    r=s1      s=s2"
$log done2
execute "GAMS tranrept    r=s2"
$log done3
```

© 2022 Prof. Bruce McCarl

Note when one does this the GAMS jobs run in the background without any log reporting appearing in the process window. To get that information you must make the DOS window visible under File Options in the Execute tab. You can also use syntax like the following (**saverestar.gms**)

```
$set gamsparm "ide=%gams.ide% lo=%gams.lo% errorlog=%gams.errorlog% errmsg=1"
execute "GAMS trandata  %gamsparm%        s=s1"
execute "GAMS tranmodl  %gamsparm%  r=s1  s=s2"
execute "GAMS tranrept  %gamsparm%  r=s2"
```

The gamsparm item here is a control variable and in this particular case it is being set up to tell GAMS that the

- IDE is being used (ide=1 on a PC)
- To write the log to a file which in turn is intercepted by the IDE (lo=3 on a pc)
- To redirect a given number of error messages to that file (errorlog=99 on a pc)
- To place error messages next to the GAMS lines causing them (errmsg=1)

The syntax %gamsparm% is placing the contents of gamsparm in the calling parameters for GAMS. The syntax %gams.ide% picks up the setting of the GAMS command line parameter ide on the computer being used while %gams.lo% and %gams.errotlog% pick up the values of the lo and errorlog GAMS command line parameters.

One can also use the command SplitOption here is one wants to provisionally include the calling parameters for GAMS.

The same syntax works with $Call.

10.3 Why use save and restart?

There are seven basic reasons to use save and restart.

- Efficiency
- Output management
- Separation of code function
- Save study results
- Comparative statics analysis
- Compiled code
- Easy solve of related models

10.3.1 Increasing run efficiency

One can use save and restart to save program status after lengthy execution permitting one to quickly do more analysis or code maintenance. Suppose we have code taking four hours and want to work on the final part. Suppose for a couple of days we have been making small changes, but have found we need to wait four hours each time to observe the results of the modifications. This would clearly be unsatisfactory and frustrating. It is also avoidable.

© 2022 Prof. Bruce McCarl

One can use save and restart to save a set of work files encapsulating the 4 hours worth of results and allowing a restart beginning from just before the revision. Generally reading in the restart file takes less than a minute. This means we can revise and rerun many times in the four hours we would have been waiting. Tactically you can do this by only executing the last component and just restarting from saved results of the earlier runs as in the following where the rem prefix makes this command into a remark (comment) that is not executed during the bat file run. Naturally if alterations were to be done in the earlier code components, one would need to rerun those to update the work files. However, one does not need to rerun unless changes are made as the work files completely encapsulate the results that would arise running the earlier files.

```
rem GAMS trandata              s=s1
rem GAMS tranmodl    r=s1      s=s2
GAMS tranrept        r=s2
```

We can even test out changes in earlier components reissuing calculations (if needed under different names), adding new variables and equations and test different formulations employing models with named equations (not /ALL/).

10.3.2 Output management

GAMS modelers sometimes complain that they want a more concise output file. All the program listings, limrow/limcol output, intermediate data displays, solver output, cross-reference maps etc are not always desirable. One-way of managing this output is through save and restarts.

Suppose we add a fourth file (**trandisp.gms**) to the transportation system that contains only display statements.

```
GAMS trandata                  s=s1
GAMS tranmodl        r=s1      s=s2
GAMS tranrept        r=s2      s=s1
GAMS trandisp        r=s1  (note reuse of name)
```

with the **trandisp.gms** file looking as follows

```
OPTION DECIMALS=0;
DISPLAY MOVEMENT,COSTS;
OPTION DECIMALS=2;
DISPLAY DEMANDREP, CMOVEMENT, shipments.l;
```

This mechanism creates "to the point" LST files but allows other output to be resident in other LST files if needed. Also one can activate or deactivate certain displays and rapidly choose relevant output for a study while maintaining other potential output in the system.

10.3.3 Separation of code function

The save and restart procedure easily allows one to substitute different data sets and maintain different pieces of code in different ways. In particular, suppose the data were kept in a separate file with saves and restarts used for the model and report writing. In turn, data specialists could maintain the data without disturbing the model or report writing code. Such a separation is not unique to saves and

© 2022 Prof. Bruce McCarl

restarts but is also exploitable when using include files. We can also use a separate data file structure to maintain small data sets for large models.

Suppose we illustrate code function separation by substituting in a larger data set in the bat file and automatically rerun a bigger model.

```
GAMS trandatl              s=b1
GAMS tranmodl        r=b1  s=b2
GAMS tranrept        r=b2  s=b1
GAMS trandisp        r=b1
```

Here we changed the restart file names to show they can be whatever desired and to avoid file confusion.

10.3.4 Save study results

Sometimes when doing applied modeling one finds a lot of experiments have been done and it's difficult to reconstruct the exact assumptions that were employed in a particular experiment. Save files provide a mechanism for archiving a file containing study results.

One can retain the save files which contain all the data and assumptions implicit in a program. Data items and stored solution results can be displayed revealing values present at the time the save file was constructed. Modifications can be made in report writing without having to rerun the whole study providing the appropriate data have been stored.

On the other hand, saving files means one cannot get an algebraic listing of equations, data manipulations etc. (the model can be resolved to list current equations or the parameters displayed to get current values). Nor can one recover any comments in the GAMS segments leading up to the saved file.

10.3.5 Comparative statics analysis

As discussed in the Basis chapter, saves and restarts can be used to save a base model from which further experimentation can be done retaining the advanced basis.

10.3.6 Compiled code

Those desiring not to release proprietary code can choose to employ a strategy where only save files are released from which others may restart but not see the source. Also you can optionally stop the client from displaying data or solution objects as discussed in Appendix H of the GAMS Users Guide.

10.3.7 Fast related solutions

Restarted models that are solved again a are restarting from a file which contains the results of a previous file generally solve quickly because of GAMS ability to suggest an advanced basis that improves solver performance. Thus one can execute and solve a model through its initial solve and then save a set of work files. In turn, restarted programs tend to rapidly solve alternative models or the same model under different options.

© 2022 Prof. Bruce McCarl

11 Customizing GAMS

Everyone has their own preferences about how programs should run. GAMS contains a number of options that allows one to manipulate

```
program function
```

and

```
output content.
```

These options may be changed on a problem-by-problem basis or may be set permanently. These notes discuss the ways to set such options and briefly cover the available options.

What types of options are there?
Possible command line parameters to customize
How can these options be set?
Hierarchy for customizing options

11.1 What types of options are there?

Mainly one can change command line parameters but on a permanent basis that applies to all jobs or all uses of the IDE. Furthermore, users can do further customization on a case-by-case basis as discussed elsewhere using $ commands and option statements in the GAMS language.

11.2 Possible command line parameters to customize

A complete list of parameters is in the chapter on Command Line Parameters. Perhaps the most desirable options to consider for run customization are

Charset	Causes use of extended (international) character set.
CNS	Name to be used for default CNS solver.
Dformat	Date format to use in output.
DNLP	Name to be used for default DNLP solver.
Errmsg	Controls position of error messages in echo print and through use of Errmsg=1 allows one to reposition error messages to just after error marking.
Inputdir	Input search paths. Can include several directories separated by OS specific symbols.
Inputdir1 to 40	Input search path names to be used. Default is no search path.
Libincdir	Library include directory. Used to complete a filename for $Libinclude.
Limcol	Include the first n cases for each named variable in the LST file.
Limrow	Include the first n cases for each named equation in the LST file.
Logoption	Controls destination for the Log file, used with setting of 2 to permit Unix/Linux jobs to operate in background.
LP	Name to be used for default LP solver.
MCP	Name to be used for default MCP solver.
MINLP	Name to be used for default MINLP solver.
MIP	Name to be used for default MIP solver.

© 2022 Prof. Bruce McCarl

MPEC	Name to be used for default MPEC solver.
NLP	Name to be used for default NLP solver.
Optdir	Location to look for solver option files.
Optfile	Default value for model.optfile, can be set to 1 if one always wants GAMS to look for option file.
Pagesize	Page size. If less than 30 it will be reset to the default of 60.
Pagewidth	Print width. This value should be between 72 and 32767.
Profile	Causes GAMS to include information on statement execution time and memory use in LST file allowing one to find slow or large memory using statements.
Ps	Page size. If less than 30 it will be reset to the default of 60.
Putdir	Default directory where put files will be saved. If not specified, it will be set to the work directory.
Pw	Print width. This value should be between 72 and 32767.
RMINLP	Name to be used for default RMINLP solver.
RMIP	Name to be used for default RMIP solver.
Savepoint	Save a point GDX file with the current solution.
Tformat	Time format.

I will not describe these fully here, as the GAMS Command Line Parameters chapter does an extensive job.

11.3 How can these options be set?

These options can be set in either a file in the GAMS system directory or in the IDE. The IDE overrides some changes so you will need to work with it to achieve desired final outcome. The possibilities are

	Globally	Locally
In pf= command line	X	X
In IDE	X	X
In GAMS parameter file	X	
In DOS call		X

Note if you use GAMS parameter file manipulation then you will be making changes for all users utilizing this GAMS version on this computer.

To set in command line via pf=
To set in Gmsprmxx.txt parameter file
To set in IDE

11.3.1 To set in command line via pf=

One can use the pf= command line parameter to specify a file containing customization items and then address that file. For example one thus could use

```
Gams myfile pf=c:\place\fileofcommands
```

© 2022 Prof. Bruce McCarl

Where fileofcommands is one containing entries like

```
Eolonly 1
Ps=9999
Pw=90
Errmsg=1
Eolonly 0
```

Use of the eolonly parameter allows one to place one command per line.

11.3.2 To set in Gmsprmxx.txt parameter file

After installation you get a file that looks like the black lines below and you can add to it

```
************************************************
* GAMS 2.50 Default Parameterfile for Windows 95
* GAMS Development Corp.
* Date : 20 Mar, 1998
************************************************
* entries required by CMEX, put in by GAMS.exe:
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
pw 85
ps 9999
errmsg 1
charset 1
LP OSL
MIP OSL
limrow 0
limcol 0
optfile 1
sp 1
pf="c:\program files\gams22.7\mycusomizing"
```

where the entries in red are the customizing values for the GAMS command line parameters

Parameter	Abbreviation	New setting above
Page width	pw	85 columns
Page lines	ps	9999 lines
Error message placement	errmsg	1-below error messages
International characters	charset	1-active
LP solver	LP	OSL
MIP solver	MIP	OSL
Row display	limrow	0-suppressed
Col display	limcol	0-suppressed

© 2022 Prof. Bruce McCarl

Optfile presence	optfile	1-always look for
Pf	pf	name of file
Savepoint	sp	1-save a file

The lines in red above once added to the Gmsprmxx.txt file are part of every GAMS execution on the computer. The above options can all be set in a GAMS call but it would be a long and involved one.

```
GAMS myfile pw=85 ps=9999  .. .optfile=1  …
```

However note a number of the options are overwritten by IDE options on a PC.

The Gamsprm - GAMS parameter file name depends on the operating system as follows

under Windows 95/98	**Gmsprm95.txt**
under Windows NT	**Gmsprmnt.txt**
under UNIX/LINUX	**Gmsprmun.txt**

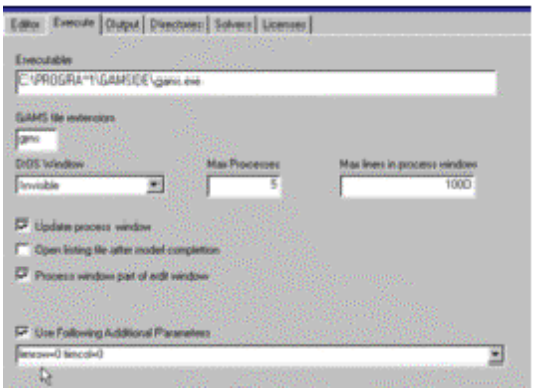
These are in the GAMS system directory that is c:\program files\GAMS22.7 as of this writing.

11.3.3 To set in IDE

One can also set these options through the IDE and many of them are retained in the ini and project files. These can be set on a global basis in two ways. First, a number of the options are accessible in the tabs under the File Options menu. Specifically under the tab for

Editor one can set tab options
Output one can control page size and width along with date and time format
Solvers one can determine default solvers

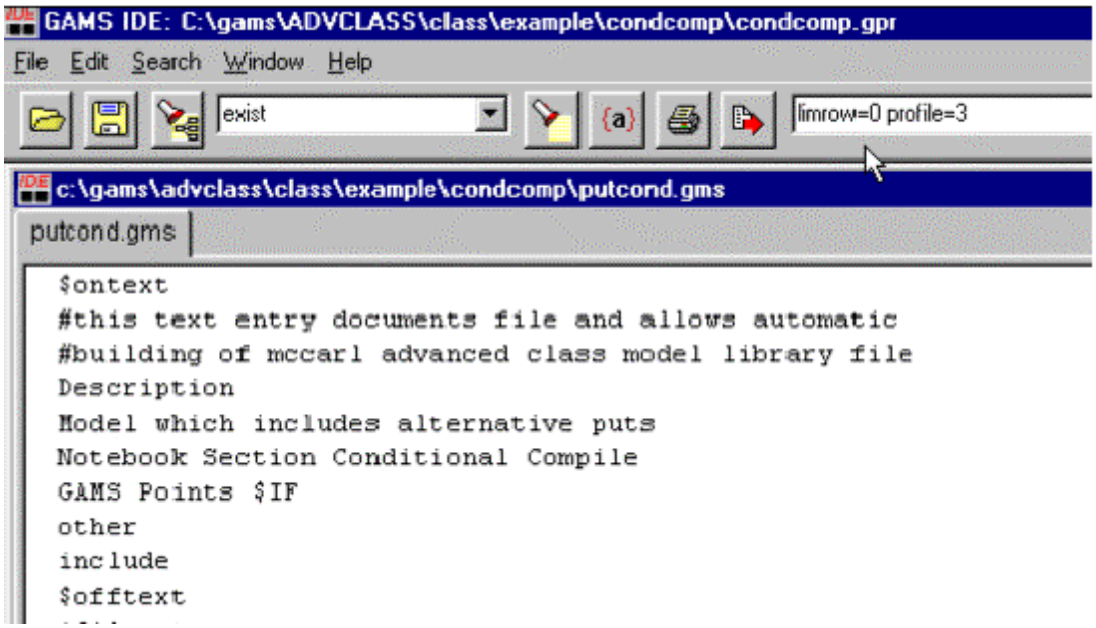
Second, for those that are not listed they can be entered in the Use Following Additional Parameters box under the Execute Tab in the File Options menu.



Finally the options can be set as parameters only for runs with particular files in the IDE by using the

© 2022 Prof. Bruce McCarl

box in the upper right hand corner.



Either of the IDE commands can be coupled with the Pf= command line parameter allowing use of a file of customization items.

11.4 Hierarchy for customizing options

One needs to be aware of the hierarchy for options when customizing GAMS on a machine or for a job as some specifications cannot be over ridden. In particular, the priorities are as follows

1. Command line specifications on GAMS command line or in IDE command line box.
2. Command line specifications in IDE additional parameters box reached by going to file, options and execute.
3. Option command specifications as discussed in the option command chapter.
4. IDE specifications for output page size and other customizable items.
5. GMSPRMxx.txt specifications.

where a higher priority setting cannot be overwritten by a lower priority setting. Thus, if one sets optfile=2 on the command line this will overcome a specification like a line stating

```
option optfile=1;
```

in the GAMS program.

11.5 List of all customizing options

The following table provides an overview of all available customizing options. The columns indicate:

© 2022 Prof. Bruce McCarl

Name	The option name
Syn	The option synonym for a cl parameter if available
Description	A description of what the option can be used for
cl	Availability as command line parameter
opt	Availability as option statement
att	Availability as model attribute
trc	Option can be echoed to trace file
api	Option is available in the Object Oriented APIs
type	Option type, e.g. 'S' for 'string', 'I' for 'integer, 'B' for 'boolean', 'IS' for 'integer or string', etc
default	Default value
lo	Lower bound
up	Upper bound

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
Action	A	GAMS processing request. R Restart After Solve C CompileOnly E ExecuteOnly CE Compile and Execute GT Trace Report	x				x	S	CE		
AppendExpand	AE	Expand file append option. 0 Reset expand file 1 Append to expand file	x				x	B	1	0	1
AppendLog	AL	Log file append option. 0 Reset log file 1 Append to logfile	x					B	0	0	1
AppendOut	AO	Output file append option. 0 Reset listing file 1 Append to listing file	x				x	B	0	0	1
AsyncSolLst		Print solution listing when asynchronous solve (Grid or Threads) is used. 0 Do not print solution listing into lst file for asynchronous solves 1 Print solution listing into lst file for asynchronous solves	x	x			x	B	0	0	1
Bratio		Basis acceptance threshold.	x	x	x		x	R	0.25	0.00	1.00
Case		Output case option for LST file. 0 Write listing file in mixed case 1 Write listing file in upper case only	x				x	B	0	0	1
CErr		Compile time error limit. 0 No error limit (default) n Stop after n errors	x				x	I	0	0	
CharSet		Character set flag. 0 Use limited GAMS characters set 1 Accept any character in comments and text items (foreign language characters)	x				x	B	1	0	1
Cheat		Cheat value, i.e. minimum solution improvement threshold.			x			R	0.00	0.00	
CNS		Constrained Nonlinear Systems - default solver.	x	x		x	x	S			
ComputerName		Computer name.				x		S			

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
CurDir	CDir	Current directory.	x					S			
CutOff		Cutoff value for branch and bound.			x			R	0.00	0.00	
Decimals		Decimal places for display statements.		x				I	3	0	8
DefPoint		Indicator for passing on default point. 0 Pass user defined levels and marginals to solver 1 Pass default levels and marginals to solver 2 Pass default marginals to solver			x			I		0	2
DFormat	DF	Date format. 0 Date as mm/dd/yy 1 Date as dd.mm.yy 2 Date as yy-mm-dy	x				x	I	0	0	2
DictFile		Force writing of a dictionary file if dictfile > 0.			x			R		0.00	
Direction		Optimization direction (min or max). 0 Minimize 1 Maximize				x		B		0	1
DispWidth		Number of characters to be printed in the column labels of all subsequent display statements.		x				I	10	10	31
DmpOpt		Debugging option: causes GAMS to echo the runtime option settings.		x				0.00	0	0	
DmpSym		Debugging option: causes GAMS to echo the symbol table to the listing file.		x				0.00	0	0	
DNLP		Non-Linear Programming with Discontinuous Derivatives - default solver.	x	x		x	x	S			
DomLim		Domain violation limit solver default.	x	x	x		x	I	0	0	
DomUsd		Number of domain violations.			x			I		0	
DualCheck		Output on the reduced cost condition.		x				I	0		
DumpOpt		Writes preprocessed input to the file input.dmp. 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	x				x	I	0	0	21
DumpParms	DP	GAMS parameter logging. 0 No logging 1 Lists accepted/set parameters 2 Log of file operations plus list of accepted/set parameters	x				x	I	0	0	2
DumpParmsLogPrefix	DPLP	Prefix of lines triggered by DumpParms>1.	x				x	S	stars		
Eject		Inject a page break into the LST file.		x				0.00	0	0	

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
EMP		Extended Mathematical Programs - default solver.	x	x		x	x	S			
EolOnly	EY	Single key-value pairs (immediate switch). 0 Any number of keys or values 1 Only one key-value pair on a line	x					I	0	0	
ErrMsg		Placing of compilation error messages. 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	x				x	I	0	0	
ErrNam		Name of error message file.	x					S			
Error		Force a compilation error with message.	x					S			
ErrorLog	ER	Max error message lines written to the log for each error. 0 No error messages to LOG file n Number of lines for each error that will be written to LOG file	x				x	I	0	0	
ETAlg		Elapsed time it took to execute the solve algorithm.			x	x		R		0.00	
ETLim	ETL	Elapsed time limit in seconds.	x				x	R	maxdouble	0.00	
ETSolve		Elapsed time it took to execute a solve statement in total.			x	x		R		0.00	
ETSolver		Elapsed time taken by the solver only.			x	x		R		0.00	
ExecErr		Execution time error limit. 0 No errors allowed limit n Max number allowed	x				x	I	0	0	
ExecMode		Limits on external programs that are allowed to be executed. 0 Everything allowed 1 Interactive shells in $call and execute commands are prohibited 2 Embedded Code and 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	x				x	I	0	0	4
Expand	EF	Expanded (include) input file name.	x				x	S			
FDDelta		Step size for finite differences.	x	x	x		x	R	1.0000000000000000E-5	1.0000000000000000E-9	1.00
FDOpt		Options for finite differences. 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	x	x	x		x	I	0	0	14

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		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		Alternative error message file.	x				x	S			
FileCase		Casing of new file names (put, gdx, ref etc.). 0 Causes GAMS to use default casing 1 Causes GAMS to upper case file names 2 Causes GAMS to lower case file names	x				x	I	0	0	3
FileStem		Sets the file stem for output files which use the input file name as stem by default.	x				x	S			
FileStemApFromEnv		Append a string read from an environment variable to the "FileStem".	x				x	S			
ForceOptFile		Overwrites other option file section mechanism.	x				x	I	0	0	
ForceWork	FW	Force GAMS systems to process save files for example with an execution error. 0 No translation 1 Try translation	x				x	B	0	0	1
ForLim		GAMS looping limit.	x	x			x	I	maxint	0	
FreeEmbeddedPython		Free external resources at the end of each embedded Python code blocks. 0 Keep resources to reuse them potentially 1 Free resources	x				x	B	0	0	1
FSave		Creates a forced \ref UG_SaveRestart "work file", i.e., the file is saved even if execution errors or other errors occured. 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	x					B	0	0	1
G205		Use GAMS version 2.05 syntax. 0 Use only latest syntax 1 Allow version 2.05 syntax only 2 Allow version 2.25 syntax only	x					I	0	0	2
GamsCloseDownTime		GAMS close down time.				x		R		0.00	
GamsCompTime		GAMS compilation time.				x		R		0.00	
GamsElapsedTime		GAMS elapsed time.				x		R		0.00	
GamsElements		GAMS elements.				x		R		0.00	
GamsErrorCount		GAMS error count.				x		R		0.00	

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
GamsExecTime		GAMS execution time.				x		R		0.00	
GamsLineNumber		GAMS number of lines.				x		R		0.00	
GamsReturnCode		GAMS return code.				x		R		0.00	
GamsStartupTime		GAMS startup time.				x		R		0.00	
GamsSymbols		Number of GAMS symbols.				x		R		0.00	
GamsTotalTime		GAMS total time.				x		R		0.00	
GamsVersionID		GAMS version ID.				x		R		0.00	
GDX		GAMS data exchange file name.	x					S			
gdxCompress		Compression of generated GDX file. 0 Do not compress GDX files 1 Compress GDX files	x				x	B	0	0	1
gdxConvert		Version of GDX files generated (for backward compatibility). v5 Version 5 GDX file, does not support compression v6 Version 6 GDX file v7 Version 7 GDX file	x				x	S	v7		
gdxUels		Unload labels or UELs to GDX either squeezed or full. Squeezed Write only the UELs to Universe, that are used by the exported symbols Full Write all UELs to Universe	x	x			x	S	squeezed		
GridDir	GDir	Grid file directory.	x				x	S			
GridScript	GScript	Grid submission script.	x				x	S	gmsgrid		
Handle		Unique handle number of SOLVE statement.			x			R		0.00	
HeapLimit	HL	Maximum Heap size allowed in MB.	x				x	R	maxdouble	0.00	
HoldFixed		Treat fixed variables as constants. 0 Fixed variables are not treated as constants 1 Fixed variables are treated as constants	x		x		x	B	0	0	1
IDE		Integrated Development Environment flag. 0 Unknown environment 1 Runs under GAMS IDE	x					B	0	0	1
Input	I	Input file.	x					S			
InputDir	IDIR	Input file directories.	x					S			
InputDir1	IDIR1	Input file directory number N.	x					S			
InputDir10	IDIR10	Input file directory number N.	x					S			
InputDir11	IDIR11	Input file directory number N.	x					S			
InputDir12	IDIR12	Input file directory number N.	x					S			
InputDir13	IDIR13	Input file directory number N.	x					S			
InputDir14	IDIR14	Input file directory number N.	x					S			
InputDir15	IDIR15	Input file directory number N.	x					S			

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
InputDir16	IDIR16	Input file directory number N.	x					S			
InputDir17	IDIR17	Input file directory number N.	x					S			
InputDir18	IDIR18	Input file directory number N.	x					S			
InputDir19	IDIR19	Input file directory number N.	x					S			
InputDir2	IDIR2	Input file directory number N.	x					S			
InputDir20	IDIR20	Input file directory number N.	x					S			
InputDir21	IDIR21	Input file directory number N.	x					S			
InputDir22	IDIR22	Input file directory number N.	x					S			
InputDir23	IDIR23	Input file directory number N.	x					S			
InputDir24	IDIR24	Input file directory number N.	x					S			
InputDir25	IDIR25	Input file directory number N.	x					S			
InputDir26	IDIR26	Input file directory number N.	x					S			
InputDir27	IDIR27	Input file directory number N.	x					S			
InputDir28	IDIR28	Input file directory number N.	x					S			
InputDir29	IDIR29	Input file directory number N.	x					S			
InputDir3	IDIR3	Input file directory number N.	x					S			
InputDir30	IDIR30	Input file directory number N.	x					S			
InputDir31	IDIR31	Input file directory number N.	x					S			
InputDir32	IDIR32	Input file directory number N.	x					S			
InputDir33	IDIR33	Input file directory number N.	x					S			
InputDir34	IDIR34	Input file directory number N.	x					S			
InputDir35	IDIR35	Input file directory number N.	x					S			
InputDir36	IDIR36	Input file directory number N.	x					S			
InputDir37	IDIR37	Input file directory number N.	x					S			
InputDir38	IDIR38	Input file directory number N.	x					S			
InputDir39	IDIR39	Input file directory number N.	x					S			
InputDir4	IDIR4	Input file directory number N.	x					S			
InputDir40	IDIR40	Input file directory number N.	x					S			
InputDir5	IDIR5	Input file directory number N.	x					S			
InputDir6	IDIR6	Input file directory number N.	x					S			
InputDir7	IDIR7	Input file directory number N.	x					S			
InputDir8	IDIR8	Input file directory number N.	x					S			
InputDir9	IDIR9	Input file directory number N.	x					S			
InputFileName		Input file name.				x		S			
Integer1		Integer communication cell N.	x	x	x		x	I		0	
Integer2		Integer communication cell N.	x	x	x		x	I		0	
Integer3		Integer communication cell N.	x	x	x		x	I		0	
Integer4		Integer communication cell N.	x	x	x		x	I		0	
Integer5		Integer communication cell N.	x	x	x		x	I		0	
InteractiveSolver		Allow solver to interact via command line input. 0 Interaction with solvelink 0 is not supported 1 Interaction with solvelink 0 is supported	x				x	B	0	0	1
IntVarUp	PF4	Set default upper bound on integer variables. 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	x	x			x	I	1	0	3

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		3 Same as 2 but issues an execution error if the level of an integer variable is greater than 100									
IterLim		Iteration limit of solver.	x	x	x		x	I	2000000000	0	
IterUsd		Number of iterations used.			x			I		0	
JobDate		Job date.				x		S	0		
JobTime		Job time in hh:mm:ss format.				x		S	0		
JobTrace	JT	Job trace string to be written to the trace file at the end of a GAMS job.	x				x	S			
JulianDate		Julian date, returns a serial day number, starting with Jan 1, 1900 as day 1.				x		S			
Keep		Controls keeping or deletion of process directory and scratch files. 0 Delete process directory 1 Keep process directory	x				x	B	0	0	1
LibIncDir	LDIR	LibInclude directory.	x				x	S			
License		Use alternative license file.	x				x	S			
LimCol		Maximum number of columns listed in one variable block.	x	x	x		x	I	3	0	
LimRow		Maximum number of rows listed in one equation block.	x	x	x		x	I	3	0	
Line		Line number of last solve of the corresponding model.			x			I		1	
LinkUsed		Integer number that indicates the value of SolveLink used for the last solve.			x			I		0	5
LogFile	LF	Log file name.	x					S			
LogLine	LL	Amount of line tracing to the log file. 0 No line tracing 1 Minimum line tracing 2 Automatic and visually pleasing	x				x	I	2	0	2
LogOption	LO	Log option. 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	x					I	1	0	4
LP		Linear Programming - default solver.	x	x		x	x	S			
LstTitleLeftAligned		Write title of LST file all left aligned. 0 Split LST title into left and right aligned part 1 Write LST title completely left aligned	x				x	B	0	0	1
Marginals		Indicator for marginals present.			x			I		0	1
MaxInfes		Maximum of infeasibilities.			x			R		0.00	
MaxProcDir		Maximum number of 225* process directories.	x				x	I	700	0	
MCP		Mixed Complementarity Problems - default solver.	x	x		x	x	S			
MCPRHoldfx		Print list of rows that are perpendicular to variables removed due to the holdfixed setting.	x	x	x			B	0	0	1
MeanInfes		Mean of infeasibilities.			x			R		0.00	
Measure		Output of time and memory use since the last		x				0.0		0	

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		measure statement or the program beginning.						0			
MemoryStat		Show memory statistics in the LST file.		x				0.00		0	
MINLP		Mixed-Integer Non-Linear Programming - default solver.	x	x		x	x	S			
MIP		Mixed-Integer Programming - default solver.	x	x		x	x	S			
MIQCP		Mixed Integer Quadratically Constrained Programs - default solver.	x	x		x	x	S			
ModelGenerationTime		Model generation time.				x		R		0.00	
ModelName		Model Name.				x		S			
ModelStat		Integer number that indicates the model status. 1 Optimal 2 Locally Optimal 3 Unbounded 4 Infeasible 5 Locally Infeasible 6 Intermediate Infeasible 7 Intermediate Nonoptimal 8 Integer Solution 9 Intermediate Non-Integer 10 Integer Infeasible 11 Licensing Problem 12 Error Unknown 13 Error No Solution 14 No Solution Returned 15 Solved Unique 16 Solved 17 Solved Singular 18 Unbounded - No Solution 19 Infeasible - No Solution			x			I		1	19
ModelStatus		Model Status.				x		I		1	19
ModelType		Model type.				x		S			
MPEC		Mathematical Programs with Equilibrium Constraints - default solver.	x	x		x	x	S			
MultiPass	MP	Multipass facility. 0 Standard compilation 1 Check-out compilation 2 As 1, and skip $call and ignore missing file errors with $include and $gdxin	x					B	0	0	2
NLP		Non-Linear Programming - default solver.	x	x		x	x	S			
NodLim		Node limit in branch and bound tree.	x		x		x	I	0	0	
NodUsd		Number of nodes used by the MIP solver.			x			I		0	
NoNewVarEqu		Triggers a compilation error when new equations or variable symbols are introduced. 0 AllowNewVarEqu 1 DoNotAllowNewVarEqu	x				x	I	0	0	
Number		Model instance serial number.			x			R			
NumberOfDiscreteVariables		Number of discrete variables (see model attribute NumDVar).				x		I		0	
NumberOfDomainViolations		Number of domain violations.				x		I		0	

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
NumberOfEquations		Number of equations (see model attribute NumEqu).				x		I		0	
NumberOfInstructions		Number of instructions.				x		I		0	
NumberOfIterations		Number of iterations.				x		I		0	
NumberOfNodes		Number of nodes.				x		I		0	
NumberOfNonlinearNonZeros		Number of nonlinear nonzeros (see model attribute NumNLNZ).				x		I		0	
NumberOfNonZeros		Number of nonzeros (see model attribute NumNZ).				x		I		0	
NumberOfVariables		Number of variables (see model attribute NumVar).				x		I		0	
NumDepnd		Number of dependencies in a CNS model.			x			I		0	
NumDVar		Number of discrete variables.			x			I		0	
NumEqu		Number of equations.			x			I		0	
NumInfes		Number of infeasibilities.			x			I		0	
NumNLIns		Number of nonlinear instructions.			x			I		0	
NumNLNZ		Number of nonlinear nonzeros.			x			I		0	
NumNOpt		Number of nonoptimalities.			x			I		0	
NumNZ		Number of nonzero entries in the model coefficient matrix.			x			I		0	
NumRedef		Number of MCP redefinitions.			x			I		0	
NumVar		Number of variables.			x			I		0	
NumVarProj		Number of bound projections during model generation.			x			I		0	
ObjectiveValue		Objective function value (see model attribute ObjVal).				x		R			
ObjectiveValueEstimate		Estimate of the best possible solution for a mixed-integer model (see model attribute ObjEst).				x		R			
ObjEst		Estimate of the best possible solution for a mixed-integer model.			x			R		0.00	
ObjVal		Objective function value.			x			R		0.00	
On115		Generate errors for unknown unique element in an equation. 0 No error messages 1 Issue error messages	x				x	B	0	0	1
OptCA		Absolute Optimality criterion solver default.	x	x	x		x	R	0.00	0.00	
OptCR		Relative Optimality criterion solver default.	x	x	x		x	R	0.10	0.00	
OptDir		Option file directory.	x				x	S			
OptFile		Default option file. 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	x		x		x	I	0	0	

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		used 1234 The option file solvername.1234 will be used									
OptionFile		Option file.				x		I		0	
Output	O	Listing file name.	x				x	S			
PageContr	PC	Output file page control option. 0 No page control, with padding 1 FORTRAN style line printer format 2 No page control, no padding 3 Formfeed character for new page	x				x	I	3	0	3
PageSize	PS	Output file page size (=0 no paging).	x				x	I	58	0	
PageWidth	PW	Output file page width.	x				x	I	255	72	32767
ParmFile	PF	Command Line Parameter include file.	x					S			
Platform		Platform.				x		S			
PLicense		Privacy license file name.	x				x	S			
PrefixLoadPath		Prepend GAMS system directory to library load path. 0 Do not set GAMS system directory at beginning of library load path 1 Set GAMS system directory at beginning of library load path	x				x	B	0	0	1
PriorOpt		Priority option for variable attribute .prior.			x			R	0.00		
ProcDir		Process Directory.	x					S			
ProcDirPath		Directory to create process directory in.	x					S			
ProcUsed		Integer number that indicates the used model type. 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			x			I		1	15
Profile		Execution profiling. 0 No profiling 1 Minimum profiling n Profiling depth for nested control structures	x	x			x	I	0	0	
ProfileFile	PFILE	Write profile information to this file.	x				x	S			
ProfileTol	PTOL	Minimum time a statement must use to appear in profile generated output.	x	x			x	R	0.00	0.00	
PutDir	PDir	Put file directory.	x				x	S			
PyMultInst		GAMS/Python Multiple Instance Interpreter.	x				x	I	0	0	1

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		0 Use a single-session Python interpreter 1 Use a multi-session Python interpreter									
PySetup		GAMS/Python Setup. 0 Rely on user setup of Python 1 Use Python installation provided in GAMS system directory	x				x	I	1	0	1
QCP		Quadratically Constrained Programs - default solver.	x	x		x	x	S			
Real1		Real communication cell N.		x	x			R			
Real2		Real communication cell N.		x	x			R			
Real3		Real communication cell N.		x	x			R			
Real4		Real communication cell N.		x	x			R			
Real5		Real communication cell N.		x	x			R			
Reference	RF	Symbol reference file.	x				x	S			
Reform		Reformulation level.		x	x			I			
ResGen		Time GAMS took to generate the model in CPU seconds(deprecated).			x			R		0.00	
ResLim		Wall-clock time limit for solver.	x	x	x		x	R	1000.00	0.00	
Restart	R	Name of a restart file, see \ref UG_SaveRestart.	x					S			
RestartNamed	RN	Name of another matching restart file, see \ref UG_SecureWorkFiles_ObfuscatedWorkFiles.	x					S			
ResUsd		Time the solver used to solve the model in CPU seconds.			x			R		0.00	
RMINLP		Relaxed Mixed-Integer Non-Linear Programming - default solver.	x	x		x	x	S			
RMIP		Relaxed Mixed-Integer Programming - default solver.	x	x		x	x	S			
RMIQCP		Relaxed Mixed Integer Quadratically Constrained Programs - default solver.	x	x		x	x	S			
RMPEC		Relaxed Mathematical Programs with Equilibrium Constraints - default solver.	x	x		x	x	S			
RObj		Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish.			x			R			
Save	S	Creates a work file, see \ref UG_SaveRestart.	x					S			
SaveObfuscate	SO	Creates an obfuscated work file, see \ref UG_SecureWorkFiles_ObfuscatedWorkFiles.	x					S			
SavePoint	SP	Save solver point in GDX file. 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	x	x	x		x	I	0	0	2
ScaleOpt		Employ user specified variable and equation scaling factors.			x			B	0	0	1
ScrDir	SD	Scratch directory.	x					S			
ScrExt	SE	Scratch file extension to be used with temporary files.	x					S	dat		
ScriptExit		Program or script to be executed at the end of a GAMS run.	x				x	S			
ScriptFrst	SF	First line to be written to GAMSNEXT file..	x					S			

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
ScrNam	SN	Work file names stem.	x					S			
Seed		Random number seed.	x	x			x	I	3141	0	
Show OSMemory		Show the memory usage reported by the Operating System instead of the internal counting. 0 Show memory by internal accounting 1 Show resident set size reported by operating system 2 Show virtual set size reported by operating system	x				x	I	0	0	2
SolPrint		Solution report print option. 0 Remove solution listings following solves 1 Include solution listings following solves 2 Suppress all solution information	x	x	x		x	IS	1	0	2
SolSlack		Causes the equation output in the listing file to contain slack variable values instead of level values. 0 includes equation levels in the solution part of the LST file following solves 1 includes equation slacks in the solution part of the LST file following solves		x				B	0	0	1
SolveLine		Line number of the solve statement.				x		I		0	
SolveLink	SL	Solver link option. 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	x	x	x		x	I	0	0	7
SolveNumber		Number of the solve statement (for the corresponding model).				x		I		1	
SolveOpt		Multiple solve management. 0 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 1 The solution information for all equations and variables is merged into the existing		x	x			IS	1	0	2

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		solution information 2 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									
Solver		Default solver for all model types that the solver is capable to process.	x	x				S			
SolverCalcTime						x		R		0.00	
SolverCntr	SCNTR	Solver control file name.	x					S			
SolverDict	SDICT	Solver dictionary file name.	x					S			
SolverElapsedTime		Elapsed solver time (slightly differs from ETSolver).				x		R		0.00	
SolverID		Solver ID.				x		I			
SolverInst	SINST	Solver instruction file name.	x					S			
SolverMatr	SMATR	Solver matrix file name.	x					S			
SolverName		Solver name.				x		S			
SolverReadTime						x		R		0.00	
SolverRealTime						x		R		0.00	
SolverSignature		e.g. IBM ILOG CPLEX 24.1.2 r40979 Released Jun 16, 2013 VS8 x86/MS Windows.				x		S			
SolverSolu	SSOLU	Solver solution file name.	x					S			
SolverStat	SSTAT	Solver status file name.	x					S			
SolverStatus		Indicates the solver termination condition (see model attribute SolveStat).				x		S			
SolverTime						x		S			
SolverVersion						x		IS			
SolverWriteTime						x		R			
SolveStat		Indicates the solver termination condition. 1 Normal Completion 2 Iteration Interrupt 3 Resource Interrupt 4 Terminated By Solver 5 Evaluation Interrupt 6 Capability Problems 7 Licensing Problems 8 User Interrupt 9 Setup Failure 10 Solver Failure 11 Internal Solver Failure 12 Solve Processing Skipped 13 System Failure			x			I		1	13
StepSum		Summary of computing resources used by job steps. 0 No step summary 1 Step summary printed	x				x	B	0	0	1
strictSingleton		Error if assignment to singleton set has multiple elements. 0 Take first record if assignment to singleton set has multiple elements	x	x			x	B	1	0	1

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		1 Error if assignment to singleton set has multiple elements									
StringChk		String substitution options. 0 No substitution if symbol undefined and no error 1 Error if symbol undefined 2 Remove entire symbol reference if undefined and no error	x				x	I	0	0	2
SubSys		Name of subsystem configuration file.	x					S			
SubSystems		Lists all solvers available as well as the current default and active solvers in the LST file.		x				0.00		0	
SumInfes		Sum of infeasibilities.			x			R		0.00	
Suppress		Compiler listing option. 0 Standard compiler listing 1 Suppress compiler listing	x				x	B	0	0	1
Symbol		Symbol table file.	x				x	S			
SymPrefix		Prefix all symbols encountered during compilation by the specified string in work file.	x				x	S			
Sys10		Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer. 0 Disable conversion 1 Enable conversion	x	x			x	B	0	0	1
Sys11		Dynamic resorting if indices in assignment/data statements are not in natural order. 0 Automatic optimization/restructuring of data 1 No optimization 2 Always optimize/restructure	x	x			x	I	0	0	2
Sys12	noSolveSkip	Pass model with generation errors to solver.	x	x			x	I	0	0	1
Sys15		Automatic switching of data structures used in search records. 0 Automatic switching to dense data structures 1 No switching 2 Always switch 1x Print additional information in lst file	x					I	0	0	
Sys16		Disable search record memory (aka execute this as pre-GAMS 24.5).	x					I	0	0	1
Sys17		Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5).	x					I	0	0	1
SysDir		GAMS system directory where GAMS executables reside.	x					S			
SysIdent		Solver identification number.			x			R			
SysIncDir	SDIR	SysInclude directory.	x				x	S			
SysOut		Solver Status file reporting option. 0 Suppress additional solver generated output 1 Include additional solver generated output	x	x	x		x	BS	0	0	1
SysVer		Solver version.			x			R			
TabIn		Tab spacing. 0 Tabs are not allowed	x				x	I	8	0	

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
		1 Tabs are replaced by blanks n Tabs are 1, n+1, 2n+1,.. (default: n=8)									
TFormat	TF	Time format. 0 Time as hh:mm:ss 1 Time as hh.mm.ss	x				x	B	0	0	1
Threads		Number of threads to be used by a solver. 0 Use number of available processors n Use n threads -n Number of processors to leave free for other tasks	x	x	x		x	I	1	minint	
ThreadsAsync		Limit on number of threads to be used for asynchronous solves (solveLink=6). 0 Use number of available processors n Use n threads -n Number of processors to leave free for other tasks	x	x			x	I	-1	minint	
Timer		Instruction timer threshold in milli seconds. 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	x				x	I	0	0	
TolInfeas		Infeasibility tolerance for an empty row of the form a.. 0*x =e= 0.0001;.			x			R		0.00	
TolInfRep		This attribute sets the tolerance for marking infeasible in the equation listing.			x			R	1.0000000000000000E-6		
TolProj		Tolerance for setting a variable level to its bound and filtering marginals when reading a solution.			x			R	1.0000000000000000E-8	0.00	
Trace		Trace file name.	x				x	S			
TraceLevel	TL	Solvestat threshold used in conjunction with action=GT.	x				x	I	0	0	
TraceOpt		Trace file format option. 0 Solver and GAMS step trace without headers 1 Solver and GAMS step trace 2 Solver step trace only 3 Trace file format used for GAMS performance world 5 Trace file with all available trace fields	x				x	I	0	0	5
TryInt		Whether solver should make use of a partial integer-feasible solution.			x			R			
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.			x			R			
User1	U1	User string N.	x			x	x	S			

© 2022 Prof. Bruce McCarl

Name	Syn	Description	cl	opt	att	trc	api	type	Default	lo	up
User2	U2	User string N.	x			x	x	S			
User3	U3	User string N.	x			x	x	S			
User4	U4	User string N.	x			x	x	S			
User5	U5	User string N.	x			x	x	S			
UserName		User name.				x		S			
Warnings		Number of warnings permitted before a run terminates.	x				x	I	maxint	0	
WorkDir	WDir	Working directory.	x					S			
WorkFactor		Memory Estimate multiplier for some solvers.	x		x		x	R	1.00	0.00	
WorkSpace		Work space for some solvers in MB.	x		x		x	R		0.00	
XSave	XS	Creates a compressed \ref UG_SaveRestart "work file".	x					S			
XSaveObfuscate	XSO	Creates a compressed \ref UG_SecureWorkFiles_ObfuscatedWorkFiles "obfuscated work file".	x					S			
ZeroRes		The results of certain operations will be set to zero if abs(result) LE ZeroRes.	x				x	R	0.00	0.00	
ZeroResRep		Report underflow as a warning when abs(results) LE ZeroRes and result set to zero. 0 No warning when a rounding occurs because of ZeroRes 1 Issue warnings whenever a rounding occurs because of ZeroRes	x				x	B	0	0	1

12 Finding and Fixing Errors or Performance Problems

This section covers the repair of computational problems that can arise when GAMS models are run, particularly large ones. The coverage is organized by chapter with the chapters covering:

Fixing Execution Errors
Scaling GAMS Models
Small to Large: Aid in Development and Debugging
Speeding up GAMS
Memory Use Reduction in GAMS

12.1 Fixing Execution Errors

The execution of a GAMS program passes through a number of stages, one of which is the execution step. During execution, errors can be detected. Generally these occur during

GAMS execution
Model generation or
Solver execution.

© 2022 Prof. Bruce McCarl

Generally, the cause of such problems is

Job requirements in excess of GAMS limits
Technical arithmetic problems where an illegal operation is performed
Critical model structure flaws

These notes cover the process of finding and fixing GAMS execution errors.

GAMS limit errors
Arithmetic errors during GAMS execution
Execution errors during model generation
Execution errors during model solution
Basing conditionals on number of errors
Clearing error conditions

12.1.1 GAMS limit errors

GAMS generates errors when it runs out of time, authorized iterations or space either during the GAMS execution of the problem or inside the solver. In turn, the LOG file can contain messages such as

```
--- Starting compilation
--- AGRESTE.GMS(312) 1 Mb
--- Starting execution
--- AGRESTE.GMS(307) 1 Mb
--- Generating model agreste
--- AGRESTE.GMS(309) 1 Mb
---    52 rows, 101 columns, and 665 non-zeroes.
--- Executing BDMLP

 BDMLP 1.3     Mar 21, 2001 WIN.BD.NA 20.0 056.043.039.WAT

 Reading data...
 Work space allocated           --    0.06 Mb

    Iter    Sinf/Objective  Status    Num  Freq
       1    5.95393922E+02  infeas      6     1
      20    2.12198277E-02  infeas      1
 SOLVER STATUS: 3 RESOURCE INTERRUPT
 MODEL STATUS : 6 INTERMEDIATE INFEASIBLE

**** SOLVER STATUS     3 RESOURCE INTERRUPT
**** MODEL STATUS      6 INTERMEDIATE INFEASIBLE
**** OBJECTIVE VALUE                0.0000
```

© 2022 Prof. Bruce McCarl

- Resources interrupt which indicates GAMS runs out of authorized time which is fixed by expanding the resource limit –

```
OPTION RESLIM = 50000;
```

  where the 50000 may be replaced with any number. Etlim can also be used.

- Too many iterations which is fixed by expanding the iteration limit –

```
OPTION ITERLIM = 100000;
```

  where the 100000 may be replaced with any number. As of 23.1 the default iteration limit (`IterLim`) was been increased from 10000 to 2e9. Setting `IterLim` to `INF` will not work since it is treated as an integer by GAMS and many solvers. Some solver, e.g. GAMS/Gurobi, recognize 2e9 and set the solver iteration limit to infinity

- Not enough work space which is fixed by expanding the workspace limit –

```
OPTION work = 30;
```

  where the number (30) gives the Work space limit in megabytes and should not exceed the computer RAM.

12.1.2 Arithmetic errors during GAMS execution

Execution time numerical errors can arise during GAMS execution calculations. These occur because of improper exponentiation (such as raising a negative number to a real power), logs of negative numbers, or dividing by zero. Such errors are marked in the LST file with **** and an associated brief message. That message indicates the nature of the arithmetic problem and the line number of the source code where the problem was encountered. Consider the example **executcl.gms**

```
sets elements /s1*s25/
parameter data1(elements)   data to be exponentiated
          datadiv(elements) divisors;
data1(elements)=1;
data1("s20")=1;
datadiv(elements)=1;
datadiv("s21")=0;
parameter result(elements);
result(elements)=data1(elements)**2.1/datadiv(elements)
display result;
```

Running GAMS we get LOG file contents of

```
--- Starting execution
--- EXECUTCL.GMS(10) 134 Kb
*** ExecError 10 at Line 10
    ILLEGAL ARGUMENTS IN ** OPERATION
```

© 2022 Prof. Bruce McCarl

```
*** ExecError 0 at Line 10
    DIVISION BY ZERO
--- EXECUTCL.GMS(11) 134 Kb 2 Errors
*** Status: Execution error(s)
--- Erasing scratch files
Exit code = 3
```

Sometimes the message indicates exactly where the problem is and the user can easily figure out the cause. Other times the problem may arise within a multidimensional item. In such cases the best way to find out where such difficulties appear is to use a display statement and show the result of the operation, then look for bad elements. This may also involve displays of the input data to the calculations so one can investigate the numerical properties of elements entered into the calculation that yields the error. Often even more displays will be involved where one needs to trace faulty input data back through the program. One would display these items repeatedly investigating places where these data have been calculated. Eventually one will find why these data have taken on the specific numerical values they have.

In the case of the above example the display of Result in the LST file reveals

```
**** EXECUTION ERROR 10 AT LINE 10 .. ILLEGAL ARGUMENTS IN ** OPERATION
**** EXECUTION ERROR 0 AT LINE 10 .. DIVISION BY ZERO

----  11 PARAMETER RESULT

s1  1.000,    s2  1.000,    s3  1.000,    s4  1.000,    s5  1.000,    s6  1.000
s7  1.000,    s8  1.000,    s9  1.000,    s10 1.000,    s11 1.000,    s12 1.000
s13 1.000,    s14 1.000,    s15 1.000,    s16 1.000,    s17 1.000,    s18 1.000
s19 1.000,    s20  UNDF,    s21  UNDF,    s22 1.000,    s23 1.000,    s24 1.000
s25 1.000
```

Here we have an indication of where the execution errors arise in the result calculation:

> For the "S20" element where we are exponentiating a negative constant to a real power raw data investigations or displays of data1 and data2 would indicate the cause is the presence in data1 of an element equal to –1 and its subsequent exponentiation.
>
> For the element "S21" where we are dividing by zero raw data investigations or displays of data1 and data2 would indicate the cause is the presence in data2 of an element equal to 0 and its subsequent use as a divisor.

So we then fix with data revisions or conditionals stopping use of data items that will cause the program to blow up.

12.1.3 Execution errors during model generation

Execution errors may arise when GAMS is generating the model before passing it to the solver. These again can be arithmetic errors or can also be model structure errors. Such model structure errors arise because some equations are improperly set up i.e. being inherently infeasible or the wrong solver is being used.

Discovery of such execution errors is sometimes very straightforward but can at other times be fairly

© 2022 Prof. Bruce McCarl

involved. An error in the middle of a multi-dimensional equation block and/or in a multi-dimensional equation term within a block can be difficult to find. The most practical way of finding such errors is to use the Limcol/Limrow option commands.

Consider the example **executmd.gms**

```
sets elems /s1*s25/
parameter data1(elems)   data to be exponentiated
          datadiv(elems) divisors
          datamult(elems)     x limits;
data1(elems)=1;
data1("s20")=-1;
datadiv(elems)=1;
datadiv("s21")=0;
datamult(elems)=1;
datamult("s22")=0;
positive variables x(elems) variables
variables  obj;
equations  objr        objective with bad exponentiation
           xlim(elems) constraints with bad divisor;
objr.. obj=e=sum(elems,data1(elems)**2.1*x(elems));
xlim(elems).. datamult(elems)/datadiv(elems)*x(elems)=e=1;
model executerr /all/
option limrow=30; option limcol=30;
solve executerr using lp maximizing obj;
```

In this example, we will have execution errors arise during model generation because of arithmetic problems. Namely

- In the objective function for the "S20" element where we are exponentiating a negative constant to a real power (because of the assignment in line 9);
- In the XLIM constraint associated with element "S21" where we are dividing by zero (because of the assignment in line 11); and
- In the XLIM "S22" constraint where we set zero equal to one which results in an infeasible constraint (because of the assignment in line 15).

When we run GAMS, we see the execution error messages as follows:

```
--- Generating model EXECUTERR
--- EXECUTMD.GMS(16) 134 Kb
*** ExecError 10 at Line 16
    ILLEGAL ARGUMENTS IN ** OPERATION
--- EXECUTMD.GMS(17) 134 Kb 1 Errors
*** ExecError 0 at Line 17
    DIVISION BY ZERO
*** ExecError 28 at Line 17
    EQUATION INFEASIBLE DUE TO RHS VALUE
--- EXECUTMD.GMS(20) 134 Kb 3 Errors
```

© 2022 Prof. Bruce McCarl

```
*** SOLVE aborted
*** Status: Execution error(s)
```

The Limrow section of the LST file reveals

```
**** EXECUTION ERROR 10 AT LINE 16 .. ILLEGAL ARGUMENTS IN ** OPERATION
---- OBJR        =E=  objective with bad exponentiation
OBJR..  - X(s1) - X(s2) - X(s3) - X(s4) - X(s5) - X(s6) - X(s7) - X(s8)
      - X(s9) - X(s10) - X(s11) - X(s12) - X(s13) - X(s14) - X(s15) - X(s16)
      - X(s17) - X(s18) - X(s19) + UNDF*X(s20) - X(s21) - X(s22) - X(s23)
      - X(s24) - X(s25) + OBJ =E= UNDF ; (LHS = UNDF, INFES = UNDF ***)

**** EXECUTION ERROR 0 AT LINE 17 .. DIVISION BY ZERO
**** EXECUTION ERROR 28 AT LINE 17 .. EQUATION INFEASIBLE DUE TO RHS VALUE

**** INFEASIBLE EQUATIONS ...
---- XLIM        =E=  constraints with bad divisor
XLIM(s22)..  0 =E= 1 ; (LHS = 0, INFES = 1 ***)
```

This does not display the equation limrow listing for the divide by zero error. To get it you have to fix the infeasibility and run again. Then you get

```
XLIM(s21)..  UNDF*X(s21) =E= UNDF ; (LHS = UNDF, INFES = UNDF ***)
```

Regardless the Limrow display shows the exact elements within the model where the problems have arisen and one may then investigate. Such an investigation may involve displays of the input data used within the equation calculations so one can investigate the numerical properties of specific elements associated with problems. Often even more displays will be involved where one traces faulty input data back through the program investigating places where these data have been calculated to eventually see why these data have taken on the specific numerical values they have.

12.1.4 Execution errors during model solution

Execution errors during model solution are generally

Arithmetic errors or

Problems caused by a presolve.

Arithmetic errors are numerically based and caused by

- Improper exponentiation (such as raising a negative value of a variable to a real power),
- Logs or square roots of negative variables,
- Squaring a negative term with **2 (use sqr or a multiplication instead of **2) and

© 2022 Prof. Bruce McCarl

- Dividing by zero.

Presolve errors either

- Indicate infeasibility or unboundedness
- Result from an overzealous presolve which eliminates the problem for all practical purposes (this generally only occurs with very small problems).

Solver function evaluation errors
Presolve errors
Solver specific limits

12.1.4.1 Solver function evaluation errors

Solvers can signal out errors using rather obscure LOG file error messages such as those below when numerical underflows or overflows are encountered during the evaluation of user model defined nonlinear terms or their derivatives. The examples below arose from a run of the problem **solvelog.gms** which has a log term in the objective function and allows the decision variable to become zero.

12.1.4.1.1 Symptoms

The messages are solver dependent. In this case MINOS yields the LOG file error message

```
EXIT -- Termination requested by User in subroutine FUNOBJ
```

while CONOPT leads to the message:

```
**    Domain error(s) in nonlinear functions.
     Check bounds on variables.
```

In turn both cause GAMS to include the error message following in the LST file:

```
****  ERRORS(S) IN EQUATION R1
1 INSTANCE OF - UNDEFINED LOG OPERATION (RETURNED -0.1E+05)
```

Essentially the same messages occur with the other types of numerical problems. For example, a model that permits division by zero is given in **solvdiv.gms** and a run of it causes similar error messages:

When using CONOPT the LOG file shows

```
** Domain error(s) in nonlinear functions.
    Check bounds on variables.
```

and the LST file shows

© 2022 Prof. Bruce McCarl

```
**** SOLVER STATUS     5 EVALUATION ERROR LIMIT
**** MODEL STATUS      6 INTERMEDIATE INFEASIBLE

**** ERRORS(S) IN EQUATION r1
     1 instance(s) of - DIVISION BY ZERO

                        LOWER     LEVEL     UPPER    MARGINAL

---- EQU r1               .       -0.400      .         EPS  INFES

                        LOWER     LEVEL     UPPER    MARGINAL

---- VAR x                .         .        +INF       EPS
---- VAR z               -INF     -0.400     +INF       EPS
```

When using MINOS the LOG file shows

```
 EXIT - Function evaluation error limit exceeded.
```

and the LST file shows

```
**** SOLVER STATUS     5 EVALUATION ERROR LIMIT
**** MODEL STATUS      7 FEASIBLE SOLUTION

**** ERRORS(S) IN EQUATION r1
     4 instance(s) of - DIVISION BY ZERO (RESULT SET TO  0.1E+05)

                        LOWER     LEVEL     UPPER    MARGINAL

---- EQU r1               .         .         .        1.000

                        LOWER     LEVEL     UPPER    MARGINAL

---- VAR x                .         .        +INF  -1.000E+4  NOPT
---- VAR z               -INF  -1.000E+4     +INF       .
```

12.1.4.1.2 Allowing errors to occur

One may allow a number of such errors to be tolerated by setting the option command Domlim.

12.1.4.1.3 Repair

Collectively these examples show that the solvers give a message indicating the type of arithmetic problem encountered and the equation where found, but do not identify the particular offending variable or the exact index case of the equation. If the cause is not obvious, one needs to investigate the numerical properties of elements within the model equations. This may involve

- Display of input data items to the nonlinear terms in the suspect equations.
- Examination of the solution report to find the markers as to where equations are infeasible

© 2022 Prof. Bruce McCarl

(INFES), variables are nonoptimal (NOPT) to see where problems are present and what variables were being manipulated at the end of the run.

- Investigation of solution items that are zero, negative or very large to see variable and equation levels at the end of the run.
- One may also need to deactivate part of the code to narrow down the problem as discussed in the notes on finding execution speed problems.

The solutions to such problems are generally

1. Add lower bounds to the problem to keep variables away from zero (x.lo(index)=0.00001).
2. Add upper bounds to the problem to keep variables away from large values (x.up(index) =1000.).
3. Reformulate the problem revising the terms and equations.
4. Provide a better starting point that keeps the solver search in a more relevant region (x.l(index) =k).
5. Fix faulty input data.

12.1.4.2 Presolve errors

Presolves in some GAMS solvers can signal execution errors problems (e.g. OSL, CPLEX, CONOPT). Circumstances where we have seen errors arise.

- Presolve can, in relatively simple problems, essentially eliminate the problem. This generally occurs because presolves commonly substitute away bounds and equality constraints to simplify the problem and may in effect simplify the problem out of existence.
- Presolve may detect the problem is unbounded or infeasible and terminate.
- CONOPT may investigate the problem and stop citing scaling inadequacies.

In such cases, the LST file solution is often mysterious and unusual. Each is illustrated below.

12.1.4.2.1 Problem eliminated

Presolve can, in relatively simple problems, essentially eliminate the problem. This generally occurs because presolves commonly substitute away bounds and equality constraints to simplify the problem and may in effect simplify the problem out of existence.

Consider the example **presol1.gms**

```
variables z;
positive variables y1,y2;
equations r1,r2,r3,r4;
  r1..  z=e=y1+y2;
  r2..  y1=l=10;
  r3..  y2=l=10;
  r4..  y1+y2=e=10;
model badpresol /all/
option lp=osl;
solve badpresol using lp maximizing z;
```

© 2022 Prof. Bruce McCarl

The LOG file reports

```
Presolve...
 **** PRESOLVE has deleted all rows

      0              0.000000     Unknown
                                  Infeasible
```

and the LST file

```
     S O L V E      S U M M A R Y
     MODEL   BADPRESOL           OBJECTIVE  Z
     TYPE    LP                  DIRECTION  MAXIMIZE
     SOLVER  OSL                 FROM LINE  10
**** SOLVER STATUS     ERROR SOLVER FAILURE
**** MODEL STATUS      6 INTERMEDIATE INFEASIBLE
**** OBJECTIVE VALUE                0.0000
```

and later we see

```
**** PRESOLVE has deleted all rows
```

The OSL presolve has made constraints r2 and r3 into simple upper bounds and has manipulated constraint r4 to express y1 in terms of y2 and y1 has been substituted out of the problem. The resultant model has one variable and no explicit constraints. In turn, the OSL solver cannot function. One would have to suppress the presolve with the OSL.opt solver option file to solve the model. The other solvers with presolves - CPLEX, XA , XPRESS and CONOPT - all can handle this.

12.1.4.2.2 No feasible mixed integer solution

Presolves can cause unusual behavior when there is no feasible integer solution. A mixed integer programming problem (**presol2.gms**) was tried which did not have a feasible integer solution.

```
variables z;
integer variables y1,y2;
equations r1,r2,r3,r4;
   r1..  z=e=y1+y2;
   r2..  y1=g=0.10;
   r3..  y2=g=0.10;
   r4..  y1+y2=l=1;
model badpresol /all/
option mip=cplex;
solve badpresol using mip maximizing z;
```

Solving this problem with CPLEX yielded the solution messages

© 2022 Prof. Bruce McCarl

```
**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      10 INTEGER INFEASIBLE
```

followed by

```
Presolve found MIP to be infeasible or unbounded.
Switching to RMIP in hopes LP solution will help with diagnosis.
Problem is integer infeasible
```

and no solution report. XA, XPRESS and BDMLP all terminate in a similar fashion although some do return a solution report.

The OSL presolve did not discover problems and cratered.

```
Presolve...
 Crashing...
 Primal Simplex...
    Iter             Objective   Sum Infeasibilities
 **** Severe problem in OSL. Check the listing file.
      Terminating...
--- Restarting execution
--- PRESOL2.GMS(14) 0 Mb
--- Reading solution for model badpresol
--- PRESOL2.GMS(14) 0 Mb 1 Error
*** Status: Execution error(s)
```

In these cases, one needs to investigate the LST file to find the messages from the presolve processor to see what happened. Also one may need to suppress the presolve using a solver options file.

12.1.4.2.3 No feasible continuous solution

Presolves can also cause unusual behavior when there is no feasible solution to the constraints. A linear programming problem (**presol3.gms**) was tried which did not have a feasible integer solution.

```
variables z;
positive variables y1,y2;
equations r1,r2,r3,r4;
  r1..  z=e=y1+y2;
  r2..  y1=g=1.10;
  r3..  y2=g=0.10;
  r4..  y1+y2=l=1;
model badpresol /all/
solve badpresol using lp maximizing z;
```

CPLEX, and OSL had experiences roughly like the following

© 2022 Prof. Bruce McCarl

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

     MODEL   badpresol           OBJECTIVE  z
     TYPE    LP                  DIRECTION  MAXIMIZE
     SOLVER  CPLEX               FROM LINE  15

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      19 INFEASIBLE - NO SOLUTION

 RESOURCE USAGE, LIMIT          0.060      1000.000
 ITERATION COUNT, LIMIT         0         10000

GAMS/CPLEX    Mar 21, 2001 WIN.CP.NA 20.0 019.019.039.WAT For CPLEX 7.0
CPLEX 7.0.0, GAMS Link 19

Unable to open options file: C:\GAMS\GAMSPDF\CPLEX.OPT.
Unable to process options file.
Presolve found the problem infeasible or unbounded.
```

and did not return a solution report. The other solvers tried (XA and XPRESS) turned off the presolve and gave back a report with some information on where the infeasibility exists.

12.1.4.3 Solver specific limits

Many solvers contain internal limits that may be exceeded and cause a LST file report of an execution errors. These are fixed by using the option commands above or a solver specific option file. Generally the LST file will tell one what options to employ. The solver manuals distributed with GAMS give the types of options that can be specified for each solver. For example to relax the MINOS major iteration limit the user should employ the file minos.opt containing the line

```
Major iterations 1000
```

or some other appropriate value.

12.1.5 Basing conditionals on number of errors

One may set up a procedure to do certain things in a job depending on the number of execution errors encountered. This is done using the function execerror. In particular one could use GAMS code as follows (**executcl.gms**)

```
result(elements)=data1(elements)**2.1/datadiv(elements);
display result;
*cause z to be undefined
scalar z;
z=1/0;
if(execerror gt 0,
   result(elements)$(result(elements) = z)=0;);
display result;
```

© 2022 Prof. Bruce McCarl

which removes the execution errors from the result array.

12.1.6 Clearing error conditions

One may also use the function Execerror to reset the count of the number of execution errors. Typically one would choose to set this to zero so GAMS terminates normally. This is done as follows (**executcl2.gms**)

```
execerror=0;
```

which causes GAMS to terminate with a normal completion message as opposed to a error code (see **executcl.gms**).

12.2 Scaling GAMS Models

Model solutions within GAMS frequently require manipulation of large matrices and many computations. The heart of most solvers includes a many numerical procedures such as a sparse matrix inverter and sets of convergence and infeasibility tolerances. Numerical problems often arise within such procedures. Poorly scaled models can cause excessive time to be taken in solving or can cause the solver to fail. GAMS can assist the user in scaling as discussed here.

Basics
Theory of scaling
Simultaneous equation and variable scaling
Scaling of GAMS models
Using GAMS scaling assistance
Effect of scaling on GAMS output
How do you know how much to scale?
A caution when scaling – runaway cases
User defined data scaling
Nonlinear scaling

12.2.1 Basics

Scaling efforts involve attempts to reduce the disparity between the coefficient magnitudes in the model including in the nonlinear variable gradients. Scaling endeavors in general try to reduce the disparity between the coefficients including the nonlinear term gradients so that the absolute value of their magnitude is centered on one and differs by no more than a multiplicative factor of no more than 1000 to 10000. Generally a well scaled model would exhibit the characteristics that

- Solution level values for the variables fall into a range around 1, e.g. from 0.01 to 100.
- Solution values of the nonzero constraint marginals exhibit absolute values falling into a range around 1, e.g. ranging in absolute value from 0.01 to 100.
- Derivatives of nonlinear terms (Jacobian elements) in the model equations fall in absolute value around 1, e.g. ranging in absolute value from 0.01 to 100 both at the starting values and at the optimal solution.
- Constants in the model equations exhibit absolute values around 1, e.g. ranging in absolute

© 2022 Prof. Bruce McCarl

value from 0.01 to 100.

To achieve such ranges the model needs to be manipulated numerically by changing the units of the variables and equations. GAMS users can employ three mechanisms that will cause such manipulations (commonly called scaling) to explicitly or implicitly occur

- Allow the solvers attached to GAMS to scale the problem
- Cause GAMS to scale using built-in scaling features
- Prescale the input data.

Solver scaling should always be done and does not require active participation by the user, but is not always sufficient. You can always do better because you understand the problem. In short, **Context is king**.

GAMS supported, user defined model scaling should be used on numerically difficult problems. You should also scale your data into sensible units.

12.2.2 Theory of scaling

Lets consider scaling theoretically. Given the LP problem

$$\begin{array}{lrcrcl} Max & c_1X_1 & + & c_2X_2 & & \\ s.t. & a_{11}X_1 & + & a_{12}X_2 & \leq & b_1 \\ & a_{21}X_1 & + & a_{22}X_2 & \leq & b_2 \\ & X_1 & , & X_2 & \geq & 0 \end{array}$$

Scaling a variable
Scaling equations

12.2.2.1 Scaling a variable

Suppose I wish alter the units of a variable like X_1 (e.g. changing it from kilograms to metric tons, requires multiplying all coefficients for X_1 by 1000). Thus, I substitute X_1' which is X_1 divided by the scaling factor (X_1/SC_1). I also adjust the a_{i1}'s and c_1 such that they are multiplied by the scaling factor

$$a_{i1}' = SC_1 * a_{i1} \qquad\qquad c_1' = SC_1 * c_1$$

and the problem becomes

© 2022 Prof. Bruce McCarl

$$
\begin{array}{lccccc}
Max & c_1'X_1' & + & c_2X_2 & & \\
s.t. & a_{11}'X_1' & + & a_{12}X_2 & \leq & b_1 \\
 & a_{21}'X_1' & + & a_{22}X_2 & \leq & b_2 \\
 & X_1' & , & X_2 & \geq & 0
\end{array}
$$

Notes:

- The solutions to the problems before and after scaling are mathematically related. *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/mccspr/new17.pdf* 1 and Spreen show that such scaling divides the optimal variable value by the scaling factor and multiplies the reduced cost or variable marginal by that factor.
- Note in doing such scaling that all coefficients for a variable have a common denominator in terms of units (resource use per unit of X_1) both before and after scaling, so every coefficient associated with X_1 needs to be multiplied by the same scaling factor to preserve denominator homogeneity of units.

12.2.2.2 Scaling equations

Scaling can also be done on equations. It again involves altering the units of model elements. For example, one might change the units of an equation from say acres to 1000 acres. To do that you would divide all associated coefficients by one thousand. In general equation scaling involves dividing very equation coefficient by the scaling factor (SR_1) as follows:

$$
\begin{array}{lccccc}
Max & c_1X_1 & + & c_2X_2 & & \\
s.t. & \frac{a_{11}}{SR_1}X_1 & + & \frac{a_{12}}{SR_1}X_2 & \leq & \frac{b_1}{SR_1} \\
 & a_{21}X_1 & + & a_{22}X_2 & \leq & b_2 \\
 & X_1 & , & X_2 & \geq & 0
\end{array}
$$

where SR_1 is a positive equation scaling factor.

Notes:

- The solutions to the problem before and after scaling have a mathematical relationship.*http://agecon2.tamu.edu/people/faculty/mccarl-bruce/mccspr/new17.pdf* show that such scaling divides

© 2022 Prof. Bruce McCarl

the optimal slack variable (or equivalently equation level) by the scaling factor and multiplies the optimal shadow price or equation marginal by that scaling factor.

- All coefficients for an equation have a common numerator (row resource units per variable unit) both before and after scaling, so every coefficient associated with the equation needs to be divided by the same scaling factor to maintain homogeneity of the numerator units in the equation.

12.2.3 Simultaneous equation and variable scaling

Reductions in numerical disparity across the coefficients are made when equations and variables are simultaneously scaled. If I scale all variables multiplying coefficients by SC_j , all rows dividing coefficients by SR_i and the objective dividing coefficients by SO then I get the following table from McCarl and Spreen applies. In the resultant problem the coefficients after scaling are given by the formulae

$$c'_j = c_j \times \frac{SC_j}{SO}, \quad a'_{ij} = a_{ij} \times \frac{SC_j}{SR_i}, \quad b'_i = \frac{b_i}{SR_i}$$

and the relationships between solution items before and after scaling is given by

Item	Symbol Before Scaling	Symbol After Scaling	Unscaled Value in Terms of Scaled Value	Scaled Value in Terms of Unscaled Value
Variable levels	X_j	X_j'	$X_j = X_j' * SC_j$	$X_j' = X_j / SC_j$
Slacks/equation levels	S_i	S_i'	$S_i = S_i' * SR_i$	$S_i' = S_i / SR_i$
Reduced costs/ variable marginals	$z_j - c_j$	$z_j' - c_j'$	$z_j - c_j = (z_j' - c_j') * (SO/SC_j)$	$z_j' - c_j' = (z_j - c_j) / (SO/SC_j)$
Shadow prices/equation marginals	U_i	U_i'	$U_i = U_i' * (SO/SR_i)$	$U_i' = U_i / (SO/SR_i)$
Obj.Function vlue	Z	Z '	Z = Z' * SO	Z '= Z / SO

Fortunately, GAMS and the GAMS solvers do this for us adjusting all solutions so they look as if they were never scaled. But this table does show the solutions are equivalent only differing by multiples of the scaling factors.

Example of scaling

© 2022 Prof. Bruce McCarl

12.2.3.1 Example of scaling

Suppose I have the problem below

$$
\begin{array}{lrcrcrcrcr}
Max & 10X_1 & - & 5000X_2 & - & 4000X_3 & - & 50000X_4 & & \\
s.t. & X_1 & - & 10000X_2 & - & 8000X_3 & & & \leq & 0 \\
 & & & 5X_2 & + & 4X_3 & - & 50X_4 & \leq & 0 \\
 & & & 1500X_2 & + & 2000X_3 & & & \leq & 6000 \\
 & & & 50X_2 & + & 45X_3 & & & \leq & 300 \\
 & X_1 & , & X_2 & , & X_3 & , & X_4 & \geq & 0
\end{array}
$$

To solve this problem I would not ordinarily use scaling, but the problem provides a vehicle to illustrate scaling procedures and consequences. The biggest numbers in the problem within the constraint matrix are in the first third and fourth constraint equations. Thus, suppose I convert the units of the first constraint by dividing all coefficients by 10000 and since it appears X_1 is in the same units convert the units of X_1 to 10000's of items by multiplying all coefficients under it by 10000. Simultaneously, I will divide all coefficients in the third constraint equation by 1000 and in the fourth by 50. The resultant model is

$$
\begin{array}{lrcrcrcrcr}
Max & 100000X_1 & - & 5000X_2 & - & 4000X_3 & - & 50000X_4 & & \\
s.t. & X_1 & - & X_2 & - & 0.8X_3 & & & \leq & 0 \\
 & & & 5X_2 & + & 4X_3 & - & 50X_4 & \leq & 0 \\
 & & & 1.5X_2 & + & 2X_3 & & & \leq & 6 \\
 & & & X_2 & + & 0.9X_3 & & & \leq & 6 \\
 & X_1 & , & X_2 & , & X_3 & , & X_4 & \geq & 0
\end{array}
$$

Now suppose I divide all coefficients in the X_4 column by 50 and all coefficients in the objective function by 10000. The final scaled problem then becomes

© 2022 Prof. Bruce McCarl

$$
\begin{array}{lrcrcrcrl}
Max & 10X_1 & - & 0.5X_2 & - & 0.4X_3 & - & 0.1X_4 & \\
s.t. & X_1 & - & X_2 & - & 0.8X_3 & & & \leq 0 \\
 & & & 5X_2 & + & 4X_3 & - & X_4 & \leq 0 \\
 & & & 1.5X_2 & + & 2X_3 & & & \leq 6 \\
 & & & X_2 & + & 0.9X_3 & & & \leq 6 \\
 & X_1 & , & X_2 & , & X_3 & , & X_4 & \geq 0
\end{array}
$$

The disparity in numbers is now much less and this is what I try to achieve in scaling. Furthermore when I solve the problem now in numerically simpler form I can reconstruct the solution to the original problem by simply multiplying and dividing solution items by the scaling factors using the above table.

12.2.4 Scaling of GAMS models

While the above theory tells how scaling works in the abstract, it is another matter as to how one goes about employing scaling in a GAMS exercise. Here I cover scaling by solvers, and users.

Scaling in GAMS solvers

12.2.4.1 Scaling in GAMS solvers

Most of the solvers GAMS uses when a SOLVE statement is executed will automatically scale problems and will transform the solution back to the unscaled one. Thus users usually gain the benefits of a scaling exercise without having to do anything.

- Solver based scaling is implemented where the solver determines the scaling factors. For example, equations may be divided through by the absolute value of the average coefficient in that equation. Similarly, variables are divided through by numbers derived from the absolute values of the average coefficient in their column. Such mechanical scaling is done iteratively for a number of passes alternating between variable and equation scaling.
- In turn the solvers automatically construct the unscaled solution using procedures like those in the table above so solver scaling is transparent to the user.
- Generally the user can do a better job than the solver in scaling due to an understanding of the model structure.
- Solver scaling can usually be suppressed through the solver options file. But this should not be done.
- Some solvers contain additional scaling procedures that are activated through the options file. Important optional scaling features are resident in MINOS as the example **nlpscale.gms** exploits.

```
MODEL nation /ALL/;
nation.optfile=1;
SOLVE nation USING NLP MAXIMIZING csps;
```

© 2022 Prof. Bruce McCarl

```
minos.opt
      superbasics  100
      scale nonlinear variables
      optimality tolerance 0.0000000001
```

12.2.5 Using GAMS scaling assistance

Generally the user can do a better job than the solver in scaling due to understanding of the model structure. GAMS also allow user controlled model scaling that can be employed to improve over solver based scaling. This procedure allows the user to specify scaling factors that are used to both alter the units in the model at hand resulting in a better-scaled model and descale the results. The procedure accepts a scale factor, both for variables and equations.

The syntax to enter such commands for variables and equations is:

```
variablename.scale(setdependency)=k1;
equationname.scale(setdependency)=k2;
```

where variablename is the name of a problem variable;
equationname is the name of a problem equation;
scale is a variable or equation attribute
setdependency are the associated set elements; and
k1 is a number or expression giving a number that will multiply all coefficients associated with the named variable
k2 is a number or expression giving a number that will divide all coefficients associated with the named equation

Scaling is turned off by default. Setting the model attribute modelname.scaleopt to 1 activates the scaling feature.

```
modelname.scaleopt=1;
```

where modelname is that named used in the model and solve statements.

The scaling statements that would be used in a GAMS program of the scaling example discussed above are (**scale.gms**)

```
scalemod.scaleopt=1;
obj.scale=10000;
z.scale=obj.scale;
avail.scale("r1")=10000;
x.scale("x1")= avail.scale("r1");
avail.scale("r3")=1000;
avail.scale("r4")=50;
x.scale("x4")=1/50;
```

In turn, GAMS would automatically scale the model and would present the results after reverse scaling to recover the original solution.

© 2022 Prof. Bruce McCarl

More generally in a model with variables named PRODUCTION and SALES along with equations named RESOURCES and PRODBAL one could introduce the scaling statements

```
modelname.scaleopt=1;
SCALFACTOR(ITEMS)=50;
SCALFACTOR("CARS")=100;
PRODUCTION.SCALE (ITEMS)      =     1000;
SALES.SCALE (PRODUCTS)  =     2000;
RESOURCES.SCALE (TYPES,OTHERSET)    =
      SCALFACTOR(ITEMS);
PRODBAL.SCALE ("CARS")  =       50;
```

much as in the manner of defining upper or lower bounds.

Note here I can set scales using set indexing as in normal GAMS calculations.

Why should you scale?

12.2.5.1 Why should you scale?

The argument was made above that you know more about model structure and therefore should be able to beat the solvers in scaling. In most cases the user can do simultaneous scaling of multiple rows and columns that can greatly narrow coefficient disparity. To see why lets return to the example,

$$
\begin{array}{lrcrcrcrl}
Max & 10X_1 & - & 5000X_2 & - & 4000X_3 & - & 50000X_4 & \\
s.t. & X_1 & - & 10000X_2 & - & 8000X_3 & & & \leq \quad 0 \\
 & & & 5X_2 & + & 4X_3 & - & 50X_4 & \leq \quad 0 \\
 & & & 1500X_2 & + & 2000X_3 & & & \leq \quad 6000 \\
 & & & 50X_2 & + & 45X_3 & & & \leq \quad 300 \\
 & X_1 & , & X_2 & , & X_3 & , & X_4 & \geq \quad 0
\end{array}
$$

In this model the variable X_1 is a sale variable that disposes of the products produced by X_2 and X_3 as added up in the first constraint. Thus, to preserve integrity of units, I scale X_1 and the first equation by the same factor.

This problem knowledge gives us a leg up and is why the user can usually do better. However solver-scaling procedures should also be used so to get the benefit of double scaling.

12.2.6 Effect of scaling on GAMS output

GAMS automatically descales all solution information so scaling does not affect the solution output. However, the LIMROW/LIMCOL output from GAMS displays the elements after scaling and an option in

© 2022 Prof. Bruce McCarl

GAMSCHK controls whether the data displayed are before or after scaling. The GAMSCHK DISPLAYCR, MATCHIT, BLOCKPIC and BLOCKLIST ordinarily print out information on an after scaling basis but POSTOPT is on a before scaling basis.

12.2.7 How do you know how much to scale?

In scaling the goal should be to reduce the coefficient absolute value range as discussed above so all numbers in the constraint matrix of the model are between 0.1 and 100. (Note that models with more disparate coefficients will work.) The steps to scaling I recommend follow

I. Examine the model first on a block-by-block basis where a block is all the set cases associated with each variable declared in the VARIABLES statements and each case for each equation declared in the EQUATIONS statements. For each block discover variables and/or constraints with absolute value maximum or minimum coefficients significantly departing from one. You can use LIMROW and LIMCOL but this is not very efficient. Use of **GAMSCHK** BLOCKPIC and BLOCKLIST is generally more efficient.

II. Develop positive scaling factors for the blocks that when employed will alter median coefficients to one concentrating on either variables or equations but ordinarily not both.

- For example, when all coefficients associated with a variable block exceed an absolute value of one (e.g. 5 to 50), then enter a variable scale factor to divide all coefficients so the median is close to one (e.g. 25).
- Simultaneously use problem knowledge to scale associated variable and/or equation blocks that should be in same units if at all possible (e.g. if you are scaling water available then scale the water sales variables by the same amount). If you don't use such knowledge you will not do any better than the solver.

III. Tell GAMS to scale with those factors.

IV. Examine the block scaling characteristics after scaling. If there are items with high or low scaling across entire blocks then iteratively work over the variables and equations until the coefficients converge in value.If the block scaling is all centered on one then proceed to individual variable and equation scaling. Here use the same steps as above employing LIMROW and LIMCOL, or **GAMSCHK** MATCHIT, PICTURE or DISPLAYCR to obtain data on model item scaling characteristics.

scalegck.gms provides an example.

12.2.8 A caution when scaling – runaway cases

A warning is in order when scaling of multidimensional variables and equations. Assume variable X (I,J,K,L,M,N) only appears in the model for a few of the possible simultaneous cases of I,J,K,L, M and N but that each of these sets has 10 elements. Naively entering the expression,

```
X.SCALE(I,J,K, L,M,N) = 100;
```

would attempt to put 1 million scaling factors into memory and could cause a computer memory overflow. One needs to be careful to only scale the variables that appear in the model. This is discussed further in the Memory and Speed chapters.

© 2022 Prof. Bruce McCarl

12.2.9 User defined data scaling

There is yet one other form of scaling that I recommend. Users can define their input data in a fashion that scales the model data and the answer. In particular, one should try to develop units for the input data so that the largest value expected for decision variables and shadow prices is under a million and in the thousands if possible.

Carefully consider the units for your data. For example, in US agriculture about 325 million acres are cropped and there is a 9-10 billion bushel corn crop. Thus in setting up production data I could enter land in 1000's of acres and all other resources in 1000's of units. I might also treat the corn crop in millions of bushels. The data will be simultaneously scaled so with resource endowments in 1000's then corn yields are divided by 1000. The net affect then is a corn production variable in the units of millions. Consumption statistics would need to be scaled accordingly. Money units can also be in say millions or billions of dollars.

Such data scaling generally greatly reduces the disparity of coefficients in the model.

12.2.10 Nonlinear scaling

Nonlinear terms merit special scaling efforts. Drud, in his **CONOPT** implementation, pays particular attention to scaling diagnostics. His recommendations for scaling are essentially those above and he also emphasizes scaling the model so that the gradients of all nonlinear terms so exhibit absolute values close to one at the starting point and at optimality.

Thus, in nonlinear models scaling should be done so that the absolute value of the nonlinear terms as reported back by GAMS LIMROW/LIMCOL and GAMSCHK are close to one.

Several actions are then in order

- One needs to use a starting point as close to the optimal solution point as possible.
- Use solver scaling. MINOS5, and CONOPT2 have special options controlling whether nonlinear scaling is done. CONOPT3 does default scaling.
- If you are having problems then scale the model using the user defined GAMS scaling options. I have worked with cases where MINOS5 would not solve the model unless user defined scaling was done.

12.3 Small to Large: Aid in Development and Debugging

Many GAMS users are overly impressed with how easily GAMS handles large models. PC based GAMS can be used with tens of thousands of variables and equations. Modelers often feel such a facility means they should always work on the full model. The result is often a large, sometimes extremely large, model in the early stages of model development. Debugging such large formulations is not easy.

The algebraic modeling style employed in GAMS is inherently expandable. This offers interesting possibilities in terms of the strategy that may be employed for model development and debugging which are discussed herein.

Basics
Essence of the small to large approach

© 2022 Prof. Bruce McCarl

Steps for working from small to large
Making small parts of large models

12.3.1 Basics

By its very nature the set based algebraic modeling style embodied in GAMS is expandable. One may employ the exact same algebra on different sized data sets.

Expandability in an example

12.3.1.1 Expandability in an example

Consider the basic transportation model (**transml.gms**) as follows where the set and parameter names are in brown, the data are in magenta and the model in blue.

```
SETS  PLANT    PLANT LOCATIONS  /NEWYORK  , CHICAGO , LOSANGLS /
      MARKET   DEMAND MARKETS   /MIAMI,   HOUSTON, MINEPLIS, PORTLAND/
PARAMETERS   SUPPLY(PLANT)  QUANTITY AVAILABLE AT EACH PLANT
            /NEWYORK   100, CHICAGO   275, LOSANGLS   90/
            DEMAND(MARKET)   QUANTITY REQUIRED BY DEMAND MARKET
            /MIAMI 100,HOUSTON 90,MINEPLIS 120,PORTLAND 90/;9
TABLE   DISTANCE(PLANT,MARKET)   DISTANCE FROM EACH PLANT TO EACH MARKET
                    MIAMI   HOUSTON   MINEPLIS   PORTLAND
       NEWYORK       1300     1800       1100       3600
       CHICAGO       2200     1300        700       2900
       LOSANGLS      3700     2400       2500       1100      ;
PARAMETER COST(PLANT,MARKET)    CALCULATED COST OF MOVING GOODS;
          COST(PLANT,MARKET) = 50 + 1 * DISTANCE(PLANT,MARKET);
POSITIVE VARIABLES
       SHIPMENTS(PLANT,MARKET) AMOUNT SHIPPED OVER A TRANSPORT ROUTE;
VARIABLES TCOST                TOTAL COST OF SHIPPING OVER ALL ROUTES;
EQUATIONS TCOSTEQ              TOTAL COST ACCOUNTING EQUATION
          SUPPLYEQ(PLANT)      LIMIT ON SUPPLY AVAILABLE AT A PLANT
          DEMANDEQ(MARKET)     MINIMUM REQUIREMENT AT A DEMAND MARKET;
TCOSTEQ.. TCOST =E=SUM((PLANT,MARKET), SHIPMENTS(PLANT,MARKET)*
                                        COST(PLANT,MARKET));
SUPPLYEQ(PLANT).. SUM(MARKET,SHIPMENTS(PLANT,MARKET))=L=SUPPLY(PLANT);
DEMANDEQ(MARKET)..SUM(PLANT,SHIPMENTS(PLANT,MARKET))=G=DEMAND(MARKET);
MODEL TRANSPORT /ALL/;
SOLVE TRANSPORT USING LP MINIMIZING TCOST;
PARAMETER MOVEMENT(*,*)  COMMODITY MOVEMENT;
MOVEMENT(PLANT,MARKET)=SHIPMENTS.L(PLANT,MARKET);
MOVEMENT("TOTAL",MARKET)=SUM(PLANT,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT(PLANT,"TOTAL")=SUM(MARKET,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT("TOTAL","TOTAL")=SUM(MARKET,MOVEMENT("TOTAL",MARKET));
OPTION DECIMALS=0;
DISPLAY MOVEMENT;
```

© 2022 Prof. Bruce McCarl

One can make a bigger implementation of that model (**tranlrg.gms**) as follows where the set and parameter names are in brown, the data are in red and the model in blue.

```
SETS  PLANT    PLANT LOCATIONS
             /NEWYORK  , CHICAGO , LOSANGLS , BALTIMORE , WASHINGTON
              PHILADEL , LASVEGAS,  RENO    , SEATTLE   , BOISE/
      MARKET   DEMAND MARKETS
               /MIAMI,   HOUSTON, MINEPLIS, PORTLAND,BOSTON/
PARAMETERS   SUPPLY(PLANT)  QUANTITY AVAILABLE AT EACH PLANT
               /NEWYORK   100, CHICAGO    75, LOSANGLS   90,
                BALTIMORE  80, WASHINGTON 70, PHILADEL   60,
                LASVEGAS   40, RENO       20, SEATTLE    55,
                BOISE      10/
             DEMAND(MARKET)   QUANTITY REQUIRED BY DEMAND MARKET
               /MIAMI     100, HOUSTON    90,
                MINEPLIS  120, PORTLAND   90,            BOSTON    180/;
TABLE DISTANCE(PLANT,MARKET) DISTANCE FROM EACH PLANT TO EACH MARKET
                    MIAMI   HOUSTON   MINEPLIS   PORTLAND  BOSTON
       NEWYORK       1300     1800       1100       3600     150
       CHICAGO       2200     1300        700       2900     800
       LOSANGLS      3700     2400       2500       1100    3800
       BALTIMORE     1100     1600       1200       3700     350
       WASHINGTON    1050     1550       1200       3700     400
       PHILADEL      1200     1700       1150       3650     250
       LASVEGAS      3300     2100       2300       1300    3600
       RENO          3400     2200       2200        900    3400
       SEATTLE       3700     2500       1900        250    3500
       BOISE         3500     2200       1700        450    3300      ;
PARAMETER COST(PLANT,MARKET)    CALCULATED COST OF MOVING GOODS;
          COST(PLANT,MARKET) = 50 + 1 * DISTANCE(PLANT,MARKET);
POSITIVE VARIABLES
       SHIPMENTS(PLANT,MARKET) AMOUNT SHIPPED OVER A TRANSPORT ROUTE;
VARIABLES TCOST                TOTAL COST OF SHIPPING OVER ALL ROUTES;
EQUATIONS TCOSTEQ              TOTAL COST ACCOUNTING EQUATION
          SUPPLYEQ(PLANT)      LIMIT ON SUPPLY AVAILABLE AT A PLANT
          DEMANDEQ(MARKET)     MINIMUM REQUIREMENT AT A DEMAND MARKET;
TCOSTEQ.. TCOST =E=SUM((PLANT,MARKET), SHIPMENTS(PLANT,MARKET)*
                                          COST(PLANT,MARKET));
SUPPLYEQ(PLANT).. SUM(MARKET,SHIPMENTS(PLANT,MARKET))=L=SUPPLY(PLANT);
DEMANDEQ(MARKET)..SUM(PLANT,SHIPMENTS(PLANT,MARKET))=G=DEMAND(MARKET);
MODEL TRANSPORT /ALL/;
SOLVE TRANSPORT USING LP MINIMIZING TCOST;
PARAMETER MOVEMENT(*,*)  COMMODITY MOVEMENT;
MOVEMENT(PLANT,MARKET)=SHIPMENTS.L(PLANT,MARKET);
MOVEMENT("TOTAL",MARKET)=SUM(PLANT,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT(PLANT,"TOTAL")=SUM(MARKET,SHIPMENTS.L(PLANT,MARKET));
MOVEMENT("TOTAL","TOTAL")=SUM(MARKET,MOVEMENT("TOTAL",MARKET));
OPTION DECIMALS=0;
DISPLAY MOVEMENT;
```

© 2022 Prof. Bruce McCarl

Comparing the models note the data contents differ between the small and the large models, but the set and parameter names **are identical** as are the model related algebraic statements.

In this case to make the larger model:

- The supply and demand sets were expanded to their new size.
- The supply availability and demand requirement data were expanded to cover the new supply and demand points.
- The distance table was expanded to include the new supply and demand points.

But, the data calculation, model definition, model solution and report writing sections are identical and the general structure of the data including all set and parameter names in the data section were identical with the data contents varying.

12.3.2 Essence of the small to large approach

The small to large approach is motivated by what we did not have to do in the above example. One can develop a model with a small data set and once the structure is right it can be used without change with larger data sets. Thus we have built the last transport model algebra we ever need to!

GAMS allows the same model structure, calculations and report writing to be developed, implemented, tested and debugged using a small data set instead of having to put up with the size, cumbersomeness, and speed degradation inherent in working with the large data set. Thus, the golden rule of GAMS model development and debugging is

work from small to large

One should to the extent possible develop, debug and augment model codes using a representative, but purposefully small, data set.

The larger the model the longer everything takes. This includes solution, compilation, generation, editing etc. Generally, time expands exponentially. Often frustration will result even when one is trying to find some relatively small data problems. Using a reduced data set permits intimate examination of model structure and function. After such verification, later expand to full problem context. But now what about the argument "I can't go back to the small data set".

Using a small model is psychologically difficult to do. Once a model is developed it is tempting to use the large data set rather than simplify.

Consider one of my mistakes. I have a linked agriculture - forest sector model used to do a greenhouse gas offset studies. Once I had to make small modifications to the constraints and report writing. After each modification the full model study was rerun covering 100+ scenarios taking 2½ days. I assured myself I didn't need to use a small model to test modifications (figuring I was a perfect modeler). Well I ended up running the 2½ day analysis 6 times, each time fixing a bug. Finally I went back to a small model and got things right in 2 hours. Sheer laziness and arrogance (I knew the model was right - 5 times over) that led me to not initially check things out using a small model, and lengthened the project by at least ten days.

© 2022 Prof. Bruce McCarl

12.3.3 Steps for working from small to large.

So how is this done? Here is a step-by-step recipe.

I. Set up a small data set representing the full model with all structural features, set names, parameters etc.

II. Implement all data calculations, model features and report writing calculations.

III. Exhaustively check the results of Step b.

IV. Save the small model. Then implement a full version with the full data set. In doing this create separate files for data, calculation, model definition and report writing so size independence is maintained. (Use include or save restart).

V. Test the larger model. Use the small part of large model techniques below to facilitate your work.

VI. Keep the small model alive. As additional structural features are added to the large model use it to test them.

12.3.4 Making small parts of large models

One will not always be able to do everything perfectly within the small model. One needs to judiciously develop the small data set so it has all the features of the large data set. I have found that most of the work can be done in the simpler setting.

Occasionally something happens in the full data set that I cannot reproduce in the small data set. There are almost always be peculiarities and interrelationships introduced when I go to the full data set. When I need to find a large data set only problem, I try one of the following strategies.

Save and restart to isolate problem area
Strategic sub-setting
Data reduction

Each of these is explained below.

Save and restart to isolate problem areas
Strategic sub-setting
Data reduction

12.3.4.1 Save and restart to isolate problem areas

Cases involving large data sets must be run. I work with several models that take hours to run. When I wish to add or verify code in a relatively small segment, an important strategy is to isolate it.

This permits repair and investigation without having to freshly input data, do initial calculations and solve. I do this using save and restart files. Consider the following example.

```
rem GAMS ALLOFIT.gms pw=80 s=.\t\save1
rem GAMS ASMMODEL.gms pw=80 r=.\t\save1 s=.\t\save2
```

© 2022 Prof. Bruce McCarl

```
rem GAMS ASMSOLVF.gms r=.\t\save2 s=.\t\save3
GAMS ASMREPT.gms pw=80 r=.\t\save3
```

The last command in the sequence involving **asmrept.gms** is executing report writer code. When I wish to make changes in the report writer code I often will run it by itself from the saved files or will even just run a smaller file containing the few instructions I wish to focus on.

12.3.4.2 Strategic sub-setting

When using full data sets in debugging or development, we usually narrowing focus to a few items and use subsets to facilitate this in an exercise I call strategic subsetting.

Consider modification of **tranlrg.gms** renaming overall sets plants and markets and introducing subsets plant and market (**transtrt.gms**). All tables, parameters, variables and equations are defined with supersets but model and calculations are defined in a subset.

```
SETS  PLANTs PLANT LOCATIONS /NEWYORK , CHICAGO , LOSANGLS , BALTIMORE , WASHINC
                              PHILADEL , LASVEGAS,  RENO    , SEATTLE   , BOI
      MARKETs   DEMAND MARKETS  /MIAMI,   HOUSTON, MINEPLIS, PORTLAND,BOSTON/
set plant (plants) a possibly reduced set of plants
                /newyork,chicago,losangls/
    market (MARKETs)   a possible reduced set of DEMAND MARKETS
                /MIAMI,   HOUSTON/;
* plant (plants)=yes ;market(markets)=yes;
PARAMETERS   SUPPLY (PLANTs)  QUANTITY AVAILABLE AT EACH PLANT
     /NEWYORK   100, CHICAGO    75, LOSANGLS   90,
      BALTIMORE  80, WASHINGTON 70, PHILADEL   60,
      LASVEGAS   40, RENO       20, SEATTLE    55, BOISE      10/
DEMAND (MARKETs)   QUANTITY REQUIRED BY DEMAND MARKET
        /MIAMI 100, HOUSTON    90,  MINEPLIS  120, PORTLAND   90, BOSTON    180/
TABLE DISTANCE (PLANTs, MARKETs) DISTANCE FROM EACH PLANT TO EACH MARKET
                    MIAMI   HOUSTON   MINEPLIS   PORTLAND  BOSTON
       NEWYORK      1300     1800       1100       3600     150
       CHICAGO      2200     1300        700       2900     800
       LOSANGLS     3700     2400       2500       1100    3800
       BALTIMORE    1100     1600       1200       3700     350
       RENO         3400     2200       2200        900    3400
       SEATTLE      3700     2500       1900        250    3500
       BOISE        3500     2200       1700        450    3300      ;
PARAMETER COST(PLANTs,MARKETs)    CALCULATED COST OF MOVING GOODS;
          COST (PLANT,MARKET) = 50 + 1 * DISTANCE (PLANT,MARKET);
POSITIVE VARIABLES    SHIPMENTS(PLANTs,MARKETs) AMOUNT SHIPPED OVER A ROUTE;
VARIABLES TCOST                 TOTAL COST OF SHIPPING OVER ALL ROUTES;
EQUATIONS TCOSTEQ               TOTAL COST ACCOUNTING EQUATION
          SUPPLYEQ(PLANTs)       LIMIT ON SUPPLY AVAILABLE AT A PLANT
          DEMANDEQ(MARKETs)      MINIMUM REQUIREMENT AT A DEMAND MARKET;
TCOSTEQ.. TCOST =E=SUM((PLANT,MARKET),
                        SHIPMENTS(PLANT,MARKET)*COST (PLANT,MARKET));
SUPPLYEQ (PLANT).. SUM(MARKET,SHIPMENTS(PLANT,MARKET))=L=SUPPLY(PLANT);
DEMANDEQ (MARKET)..SUM(PLANT,SHIPMENTS(PLANT,MARKET))=G=DEMAND(MARKET);
```

© 2022 Prof. Bruce McCarl

```
MODEL TRANSPORT /ALL/;
SOLVE TRANSPORT USING LP MINIMIZING TCOST;
```

In turn altering the lines defining the plant and market subsets as follows would alter problem from the restricted problem back to the full model.

```
set plant(plants) a possibly reduced set of plants
*                 /newyork,chicago,losangls/
      market(MARKETs) a possibly reduced set of MARKETS
*                 /MIAMI,   HOUSTON/
;
      plant (plants)=yes ;market(markets)=yes;
```

Furthermore, since the plant and market sets are now calculated items this could be done anywhere. Thus, if one were employing code isolation one could further narrow the things being examined using strategic sub setting.

Strategic sub setting has proven to be an effective way of maintaining a small data set with little effort. All one really does is pick elements from the full set that are representative for model development and debugging.

12.3.4.3 Data reduction

Another variant of the problem reduction strategy is to exploit the fact that GAMS skips cases where data items are zero. Thus data can be temporarily removed from data set by setting items to zero. Consider the following example (**tranzer.gms**)

```
set   origin            /o1*o100/
                destinat  /d1*d100/;
parameter distance(origin,destinat);
      distance(origin,destinat)=120+50*ord(destinat)-10*ord(origin);
set   smallorig(origin) small set of origins for testing /o4,o47,o91/
      smalldest(destinat) small set of destinations /d3,d44,d99/;
distance(origin,destinat)
     $(not (smallorig(origin) and smalldest(destinat)))=0;
parameter cost(origin,destinat);
Cost(origin,destinat)$distance(origin,destinat)
       =3+2*distance(origin,destinat);
display cost,distance;
```

Here we have zeroed most of the distances and if the rest of model were conditioned on nonzero transportation costs, then this would greatly reduce model size. This also illustrates strategic sub-setting.

12.4 Speeding up GAMS

Program execution time and memory usage is often a function of the GAMS implementation, which can be changed without substantively altering the results of the program. I have seen alteration of a few

© 2022 Prof. Bruce McCarl

statements cause huge efficiency gains to be achieved. For example, I have seen reductions in execution time from 30 minutes to 15 seconds achieved by rewriting a small amount of GAMS code without changing results. Here I cover

Diagnosis	whether and where there is a problem
Causality	features of GAMS that cause time problems to occur
Repair	manipulation of the GAMS code to repair the problem.

I limit coverage to the speed of execution within GAMS not covering manipulations to reduce time usage within a solver called by GAMS.

Basics
Finding where excessive time is being used
Why programs can be slow and their repair
Trading memory for time
Other speed ups

12.4.1 Basics

GAMS can take a lot of time in computations and model setup. When confronted with a program that takes a long time, ask yourself some questions:

- Does the program take more time than you feel it should?
- During execution does the screen show execution of one line number for a long time?
- Is the procedure used often enough that efficiency is a concern?

If the answer to any of these questions is yes, then further investigation is in order to see whether there are poorly executing portions of the program.

12.4.2 Finding where excessive time is being used

The best strategy for discovering causes of slow execution and eliminating the problem is a mixture of problem reduction and the techniques below. The problem reduction strategy will not be discussed here (see the Small to Large chapter). Beyond that the strategies one can used involve

- Tracking program execution through screen watching and LOG file / process window contents examination.
- Tracking program execution characteristics through profile and profiletol usage.
- Finding problems within large slow statements by searching and code isolation.
- Finding problems when GAMS goes on forever and you cannot wait for the profile by searching and code isolation.

Screen watching and LOG file examination
Profile

© 2022 Prof. Bruce McCarl

12.4.2.1 Screen watching and LOG file examination

One can carefully watch the screen or IDE process window (LOG file) report during execution. In that display, GAMS reports the line number that it is executing. If the program pauses on a line number for a moderately long time, then one would look at that line as a cause of slow execution (reasons why a statement may be slow are discussed below).

Example:

When I run **gamsslow.gms**, the diligent screen watcher may see that the line reporting pauses longest on statements 29,30,31,32 but I really can't on my computer.

Notes:

Screen watching and or LOG file examination are not good problem detection techniques for two reasons.

- Staring at the screen for long time periods may not be effective and one may miss certain statements, may identify statements improperly or get distracted and have to redo the approach repeatedly.
- GAMS line reporting is misleading when loops, if and other control structure statements are being executed. For example in running **gamsloop2.gms** the statements within the LOOP are all reported as if they were at the line number of the loop statement -28 in this case. Individual calculations in the loop are not reported to the screen i.e. lines 29-31. Thus through screen watching, one would not get any indication other than the loop is taking a lot of time and would not know where within the loop to look. The same thing happens within if statements and other GAMS control structures.

12.4.2.2 Profile

GAMS users can cause the output to contain information on statement execution time and associated memory usage by employing profile. Profile is invoked using an option or a command line parameter as I discuss below. When profile is activated in a model, the LST file contains profiling lines. These lines are collected and reproduced below for a run of the **gamsslow.gms** model.

```
----     13 ASSIGNMENT x             0.090     0.090 SECS    5.7 Mb  172800
----     15 ASSIGNMENT z             0.520     0.610 SECS    9.9 Mb  172800
----     17 ASSIGNMENT y             0.561     1.171 SECS    9.9 Mb
----     29 ASSIGNMENT slow          0.000     1.171 SECS    9.9 Mb
----     30 SOLVE INIT slow          0.000     1.171 SECS    9.9 Mb
----     24 EQUATION   objeq         1.142     2.313 SECS   10.5 Mb       1
----     25 EQUATION   R             0.861     3.174 SECS   17.8 Mb    1200
----     26 EQUATION   q             1.142     4.316 SECS   18.3 Mb    1440
----     30 SOLVE FINI slow          0.120     4.436 SECS   18.3 Mb
----      1           EXEC-INIT      0.000     0.000 SECS    9.2 Mb
solve appeared here
----     30 SOLVE READ slow          0.010     0.010 SECS    9.7 Mb
----     32 ASSIGNMENT sumofvar      0.701     0.711 SECS   10.5 Mb
```

The columns of this report are:

© 2022 Prof. Bruce McCarl

- A demarking string ---- flagging this line. The line also contains the text string SECS (which is usually the better thing to search for).
- GAMS statement number of the instruction being profiled.
- The item name of the GAMS symbol being worked on.
- The execution time of each statement.
- Cumulative program execution time.
- Current memory use.
- The number of cases for which the statement is executed (if the cases exceed one).

Note

- As of version 23.1 when Profile is used a summary report of the ten slowest execution steps will be written to the log and listing files.
- The timing on slow data definitions and gdx loads during compilation is profiled as well.
-

12.4.2.2.1 Use of profile to find slow statements

Now lets look at what the profile reveals.

```
----     13 ASSIGNMENT x              0.090     0.090 SECS    5.7 Mb  172800
----     15 ASSIGNMENT z              0.520     0.610 SECS    9.9 Mb  172800
----     17 ASSIGNMENT y              0.561     1.171 SECS    9.9 Mb
----     29 ASSIGNMENT slow           0.000     1.171 SECS    9.9 Mb
----     30 SOLVE INIT slow           0.000     1.171 SECS    9.9 Mb
----     24 EQUATION   objeq          1.142     2.313 SECS   10.5 Mb       1
----     25 EQUATION   R              0.861     3.174 SECS   17.8 Mb    1200
----     26 EQUATION   q              1.142     4.316 SECS   18.3 Mb    1440
----     30 SOLVE FINI slow           0.120     4.436 SECS   18.3 Mb
----      1           EXEC-INIT       0.000     0.000 SECS    9.2 Mb
solve appeared here
----     30 SOLVE READ slow           0.010     0.010 SECS    9.7 Mb
----     32 ASSIGNMENT sumofvar       0.701     0.711 SECS   10.5 Mb
```

Note the red lines identify the statements by number and symbol where large execution times are encountered (i.e., statements 15,17, 24, 25, 26, and 30). In turn, one can examine those statements to see if they can be reworked for faster execution. Reasons why statements may be slow are discussed below.

12.4.2.2.1.1 Invoking profile

Profile may be invoked in a number of different ways.

On the GAMS command line

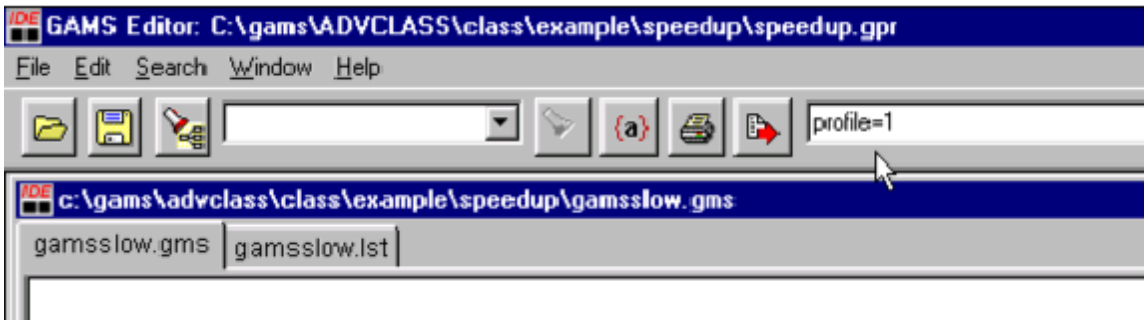
© 2022 Prof. Bruce McCarl

In the IDE GAMS parameters box
As an internal option

Profile is a valid command line option and may be set as follows

```
GAMS MYMODEL     PROFILE = 1       DOS
GAMS MYMODEL     -PROFILE 2        UNIX
```

One can enter the command line option in the IDE parameters box.



One can place an option statement into the program as follows:

```
OPTION PROFILE=3;
```

12.4.2.2.1.2 What should the number be

On entering profile it is set equal to a number

i.e.

```
option profile=1    or    option profile=3
```

The use of higher numbers causes GAMS to carry out profiling within control structures such as loop, if, repeat, for and when. The number tells how deep within the control structures to go.

If profile is set to

1 GAMS reports statement timing and memory use at the control statement level without details on statements within control structures.

2 GAMS includes output on statement timing and memory use for all statements that are not within control statements plus the first level of statements within control structures.

3 GAMS includes profile reports on statement timing and memory use for all statements that are not within control statements and on statements nested within a second level of control statements (i.e. loops or ifs within loops).

gamsloop.gms provides an example. Note profile can have values of 4 and above.

12.4.2.2.1.3 Limiting profile output: Profiletol

Profile can generate a tremendous amount of output, much of which is not informative. The above profile

© 2022 Prof. Bruce McCarl

reports show several statements reported for which there is not meaningful execution time. One can suppress this information by using a tolerance on the minimum amount of execution time in seconds that a statement must use to be reported.

```
OPTION PROFILETOL = 0.5;
```

or

```
OPTION PROFILETOL = 2;
```

or

```
OPTION PROFILETOL = 10;
```

In bigger models the latter would cause reporting of statements that took 2 or 10 or more seconds of execution time. Bigger numbers yet can be used.

12.4.2.2.2 Isolating terms in slow statements

Sometimes one runs profile and finds that the time problem is inside a tremendously long statement. For example, I have been known to regularly run models where the objective function and some report calculation lines are well over 800 lines long containing 40 or so added terms. Finding a timing problem in such a long statement still leaves me with the question of where is the problem. In such cases I often temporarily deactivate part of the code using the comment inserting syntax $Ontext/$Offtext and *'s.

Namely, if a multi-term piece of code spanning lines 1182 to 1294 is slow, one could would re-execute the code, but split the term so say lines 1240 to 1294 are deactivated (surrounding them with an $Ontext - $Offtext sequence). Then, if the remaining lines still use the bulk of the time, I would deactivate further code until the code deactivation greatly reduced the time use. I would then know that the last section put into the text status contained the slow executing code. I would in turn selectively activate more code until the exact problematic portion is identified. Naturally after finding and fixing my timing problem I would reactivate all of the temporarily commented out code.

12.4.2.2.3 It takes too long - searching

Sometimes code executes too slow to wait or to put in profile and get the output (i.e. it runs for 3 hours and shows no signs of abating). But if I abort the job the operating system buffer handing procedures generally cause loss of the last few lines of profile information when the job was aborted. So how do I find the problem? The answer is I use

The problem reduction strategies in the small to large chapter

or

Search using code deactivation and isolation

The former involves the modeling simplification strategies discussed in the small to large chapter along with the profile and memory dump techniques discussed above with the and will not be elaborated on here. The latter merits discussion.

© 2022 Prof. Bruce McCarl

When I have code that just won't work fast enough to wait, I can engage in a **quest for the last good statement**. This can be done by using code deactivation by commenting out statements using

$Ontext/$Offtext to temporarily make comments of active statements
Making individual lines into comments using *

In turn, I would successively deactivate more and more statements until the performance changes with the job terminating appropriately. I would then slowly activate members of the last code statements deactivated until the code performance got worse again. Then by iteratively activating and deactivating I can isolate the exact problematic terms. I may also use save restart to speed this up by saving the results up until a known good spot and just executing the suspect statements.

12.4.3 Why programs can be slow and their repair

There are three big reasons a program may be slow. These are

Set addressing and references
Considering unnecessary cases
Post solution report writing computations

I discuss each below, but before beginning I wish to note that the material below is not only useful for the repair of specific models. Namely, by employing the efficiency enhancing practices discussed herein in standard GAMS modeling, one can improve the effectiveness of any GAMS programs to which these techniques are applied. For example, the section on set addressing will reveal conventions that should be used in all GAMS programming yielding efficiency gains without the need for model efficiency investigations.

Set addressing and references
Avoiding considering unnecessary cases

12.4.3.1 Set addressing and references

GAMS employs a sparse matrix data storage scheme. A parameter like X(A,B,C) is stored in the order

```
a1 b1 c1
a1 b1 c2
...
a1 b1 cm
a1 b2 c1
...
a1 b2 cm
...
a1 bn cm
a2 b1 c1
```

© 2022 Prof. Bruce McCarl

```
...
ak bn cm
```

Note the entries are stored internally in systematic order with the last entry varied the fastest then the second then the first. GAMS withdraws entries from memory fastest if they are referenced in the order most consistent with the storage order. As a consequence the calculation specification

```
Y(a,b,c)=X (a,b,c);
```

is faster than

```
Y(a,b,c)=X(b,c,a).
```

Referencing in a manner inconsistent with the storage order slows things down. One should arrange set reference order so that across calculations, equation definitions etc, the sets are always referenced in the same order. We then arrive at speed tip number one. To increase speed modelers should endeavor to arrange definitions, calculations, sums, and equation references to sets in a consistent order. Consider the example (**gamsslow.gms** versus **gamsfast.gms**)

```
parameter x(e,d,c,b,a);
X(e,d,c,b,a)=10;
parameter z(a,b,c,d,e);
z(a,b,c,d,e)=x(e,d,c,b,a);
parameter y;
                Y=sum ((a,b,c,d,e),z(a,b,c,d,e)*x(e,d,c,b,a));
variables obj
Positive variables var(e,b,a);
equations objeq
          R( b,c,d)
          q(a,b,c);

objeq.. obj=e=sum((a,b,c,d,e),z(a,b,c,d,e)*x(e,d,c,b,a)*var(e,b,a));
r(b,c,d).. sum((a,e),Var(e,b,a))=l=sum((a,e),x(e,d,c,b,a)*z(a,b,c,d,e));
q(a,b,c).. sum((d,e),var(e,b,a)/x(e,d,c,b,a)*z(a,b,c,d,e))=l=20;
model slow /all/;
*option lp=bdmlp;
slow.workspace=10;
solve slow maximizing obj using lp;
parameter sumofvar;
sumofvar=sum((a,b,c,d,e),z(a,b,c,d,e)*x(e,d,c,b,a)*var.l(e,b,a));
```

which yields the Profile lines where the coloring corresponds to source file lines.

```
----     13 ASSIGNMENT x              0.090     0.090 SECS    5.7 Mb  172800
----     15 ASSIGNMENT z              0.520     0.610 SECS    9.9 Mb  172800
----     17 ASSIGNMENT y              0.561     1.171 SECS    9.9 Mb
----     29 ASSIGNMENT slow           0.000     1.171 SECS    9.9 Mb
----     30 SOLVE INIT slow           0.000     1.171 SECS    9.9 Mb
----     24 EQUATION   objeq          1.142     2.313 SECS   10.5 Mb       1
```

© 2022 Prof. Bruce McCarl

```
----     25 EQUATION   R              0.861     3.174 SECS   17.8 Mb   1200
----     26 EQUATION   q              1.142     4.316 SECS   18.3 Mb   1440
----     30 SOLVE FINI slow           0.120     4.436 SECS   18.3 Mb
----      1           EXEC-INIT       0.000     0.000 SECS    9.2 Mb
solve appeared here
----     30 SOLVE READ slow           0.010     0.010 SECS    9.7 Mb
----     32 ASSIGNMENT sumofvar       0.701     0.711 SECS   10.5 Mb
```

A rearrangement of set indexes to a consistent order in **gamsfast.gms** yields the profile information

```
----     13 ASSIGNMENT x              0.050     0.050 SECS    3.7 Mb  100000
----     15 ASSIGNMENT z              0.090     0.140 SECS    6.3 Mb  100000
----     17 ASSIGNMENT y              0.110     0.250 SECS    6.3 Mb
----     29 ASSIGNMENT slow           0.000     0.250 SECS    6.3 Mb
----     30 SOLVE INIT slow           0.000     0.250 SECS    6.3 Mb
----     24 EQUATION   objeq          0.410     0.660 SECS    6.3 Mb       1
----     25 EQUATION   R              0.401     1.061 SECS   11.0 Mb    1000
----     26 EQUATION   q              0.411     1.472 SECS   11.0 Mb    1000
----     30 SOLVE FINI slow           0.070     1.542 SECS   11.0 Mb
----     30 GAMS FINI                 0.030     1.572 SECS   11.0 Mb
----     30 SOLVE READ slow           0.010     0.010 SECS    6.1 Mb
----     32 ASSIGNMENT sumofvar       0.150     0.160 SECS    6.8 Mb
```

Where note the redefinition of the line numbered 15

```
z(a,b,c,d,e)=x(e,d,c,b,a);
```

into

```
z(a,b,c,d,e)=x(a,b,c,d,e);
```

drops execution time from 0.52 to 0.09 seconds. Substantial percentage reductions were achieved in all the time consuming cases by the reordering and improved consistency of set addressing.

12.4.3.2 Avoiding considering unnecessary cases

One way to speed up GAMS is to make sure only necessary cases are indexed. There are five contexts which merit consideration

Calculation statements
Equation existence
Equation terms
Variable specification
Post solution computations

12.4.3.2.1 Calculation statements

A calculation of a parameter that is defined over a large of number of sets such as the following can

© 2022 Prof. Bruce McCarl

inadvertently cover a huge number of cases

```
X(A, B, C, D, E) = 5
```

If each set had 20 members then the calculation would have to cover 3.2 million cases and would take a long time. This can be helped by narrowing attention only to good cases employing conditionals.

```
X(A, B, C, D, E) $ GOODCASE (A, B, C, D, E) = 5
```

where the user must somehow define the good case item, perhaps as a **tuple**.

Example:

(landcal.gms versus landexam.gms)

Suppose I have states and counties in my data set and wish to sum. In the absence of other information GAMS would consider all counties as possibly being in all states. Thus, New Jersey counties are considered in doing sums involving Texas. Consider 2 cases of the same calculation

In **landcal.gms** every county is treated as if it is in every state

```
12  landarea(state,county,landtype)=1;
13  parameter totalland(state);
14  totalland(state)=
15        sum(county,sum(landtype,landarea(state,county,landtype)));
```

which generates the profile lines

```
----     12 ASSIGNMENT landarea      0.811     0.811 SECS   39.8 Mb  1500000
----     14 ASSIGNMENT totalland     0.731     1.542 SECS   39.8 Mb     50
```

In **landexam.gms** a county is treated in a state only when the county is matched up with the state in the matchup array

```
16  landarea(state,county,landtype)$matchup(state,county)=1;
17  parameter totalland(state);
18  totalland(state)=
19        sum(matchup(state,county),sum(landtype,landarea(state,county,landtyp
```

The resultant profile is

```
----     16 ASSIGNMENT landarea      0.020     0.380 SECS    2.1 Mb  30490
----     18 ASSIGNMENT totalland     0.010     0.390 SECS    2.1 Mb     50
```

and the second version treats a lot less cases, executes in executes in about 1/4 the total time and greatly reduces memory use. In general, speed can often be helped by using tuples, subsets or conditionals to reduce attention in calculations only to relevant cases.

© 2022 Prof. Bruce McCarl

12.4.3.2.2 Equation existence limited using conditionals

(transbad.gms vs. transfix.gms)

Equations such as

```
slDEMANDEQ(MARKET)..
SUM(PLANT, SHIP(PLANT, MARKET))=G=DEMAND(MARKET);
```

may not really be relevant for all possible elements of the market set. I might only define equations when there is nonzero demand. Program execution time and model size would be helped by

```
inDEMANDEQ(MARKET)$demand(market)..
SUM(PLANT,  SHIP(PLANT, MARKET)) =G= DEMAND(MARKET);
```

In general, speed can often be helped by using tuples, subsets or conditionals to reduce attention only to relevant equations.

12.4.3.2.3 Equation term consideration limited using conditions

(transbad.gms vs. transfix.gms)

An equation such as

```
slDEMANDEQ(MARKET)..
SUM(PLANT, SHIP(PLANT, MARKET))=G=DEMAND(MARKET);
```

can be generated faster by adding conditionals both on equation and term existence to avoid considering unnecessary cases

```
fsDEMANDEQ(MARKET)$demand(market)..
SUM(PLANT $cost(plant,market),SHIP(PLANT, MARKET)) =G=
DEMAND(MARKET);
```

In general, speed can often be helped by using tuples, subsets or conditionals to reduce inclusion of terms in .. equations only to relevant cases.

12.4.3.2.4 Variable specification - suppression

(transbad.gms vs. transfix.gms)

Cases can exist where unneeded variables are being defined. Creation of an unnecessary variable requires extra execution time. For example, in a transport model, I may only want variables that are defined over routes with nonzero transportation cost.

```
TCOST =E= SUM((PLANT,MARKET)$cost(plant,market)
          , SHIPMENTS(PLANT,MARKET)*COST(PLANT,MARKET));
fsSUPPLYEQ(PLANT)$supply(plant)..
  SUM(MARKET $cost(plant,market), SHIPMENTS(PLANT, MARKET))=L=SUPPLY(PLANT);
```

© 2022 Prof. Bruce McCarl

```
fsDEMANDEQ(MARKET)$demand(market)..
  SUM(PLANT$cost(plant,market),  SHIPMENTS(PLANT, MARKET))=G=DEMAND(MARKET);
```

In general, speed can often be helped by using tuples, subsets or conditionals to reduce attention in variable inclusion across .. equations only to relevant cases.

12.4.3.2.4.1 Watch out for incomplete suppression

(trnssu.gms)

When I use conditionals to eliminate variables I must be careful to watch out for cases of incomplete elimination. Variables may still be there that I thought were gone. Below, I do not suppress zero transport cost cases in the bottom 2 equations and will end up with variables present that trivially allow demand satisfaction.

```
TCOST =E= SUM((PLANT,MARKET)$cost(plant,market)
, SHIPMENTS(PLANT,MARKET)*COST(PLANT,MARKET));
fsSUPPLYEQ(PLANT)$supply(plant)..
      SUM(MARKET, SHIPMENTS(PLANT, MARKET))=L=  SUPPLY(PLANT);
fsDEMANDEQ(MARKET)$demand(market)..
      SUM(PLANT,  SHIPMENTS(PLANT, MARKET)) =G= DEMAND(MARKET);
```

In general poor quality answers can be avoided if one is careful to suppress variables in a consistent fashion across all equations.

12.4.3.2.5 Post solution report w riting computations

Often modelers employ post solution report writing calculations. These calculations can involve retrieving and manipulating a lot of data then multiplying it by the optimal variable levels

```
Y=SUM((A,B,C,D,E,F,G),
      (DAT(A)+IT(B,C)+Y(D,E)+W(F,G))*X.L(A,B,C,D,E,F,G))
```

Such calculations will virtually always perform better if the modeler enters a conditional that only causes the data retrieval and calculations to start if the solution variable value is nonzero

```
Y=SUM((A,B,C,D,E,F,G)$X.L(A,B,C,D,E,F,G),
      (DAT(A)+IT(B,C)+Y(D,E)+W(F,G))*X.L(A,B,C,D,E,F,G));.
```

This can be a huge time saver. It relies on the fact that in general few variables will be nonzero in a programming model compared to the number of variables present.

In general, speed can often be helped by using tuples, subsets or conditionals to reduce attention in report writing calculations only to terms associated with nonzero decision variable values.

12.4.4 Trading memory for time

Sometimes an extensive calculation that is repeated in the model many times can be restructured so it

© 2022 Prof. Bruce McCarl

is calculated once then saved. For example, a component of a .. equation specification for a model solved a lot of times can be restructured to be done once and stored then only accessed later as follows.

```
obj.. Z=SUM(( CROP, TILLAGE, LANDTREAT, ROTATION ),
             ACREPLANT( CROP, TILLAGE, LANDTREAT, ROTATION)*
           SUM(INPUT, USAGE(INPUT,  CROP));
```

Here the code is revised by defining a parameter for the input usage sum and substituting i.e.;

```
INPUTUSE( CROP) = SUM(INPUT, USAGE (INPUT, CROP));
obj.. Z=SUM(( CROP, TILLAGE, LANDTREAT, ROTATION ),
  ACREPLANT( CROP, TILLAGE, LANDTREAT, ROTATION)*
      INPUTUSE (CROP));
```

This can be a huge time saver but one needs to watch out for problems with static calculations as discussed in the calculating items chapter.

12.4.5 Other speed ups

In addition to the strategies above one can try to enhance speed-using items discussed in other chapters of this document.

Scaling
Advanced Basis Usage
Starting Points

One can alto try

Changing the solver choice

or

Reformulating the problem

12.5 Memory Use Reduction in GAMS

Memory usage by a GAMS program is often a function of the implementation. Code can often be changed so memory requirements are reduced without substantively altering the bottom line results of the program. For example, I have seen substantial reductions in memory requirements achieved by rewriting a small amount of GAMS code without changing results. Here I cover

Diagnosis	whether and where there is a problem
Causality	features of GAMS that cause memory problems to occur

© 2022 Prof. Bruce McCarl

Repair	manipulation of the GAMS code to repair the problem.

I limit coverage to the memory usage within GAMS not covering manipulations to reduce memory use within a solver called by GAMS.

Basics
Finding where excessive memory is being used
Causes of excessive memory use and repair
Limiting memory usin using HeapLimit

12.5.1 Basics

GAMS can use a lot of space in computations and model setup. When confronted with a program that takes a lot of memory, ask yourself some questions:

- Does the program take more memory than you feel it should or than the computer will allow?
- Does the memory reporting on the screen show a large increase after executing of one line number?
- Is the procedure used on machines where memory limits are of concern?

If the answer to any of these questions is yes, then further investigation is in order to see whether the memory use characteristics of a GAMS program can be improved.

12.5.2 Finding where excessive memory is being used

The best strategy for discovering causes of excess memory usage and eliminating the problem involves a mixture of problem reduction and the techniques below. The problem reduction strategy will not be discussed here (see the Small to Large chapter). Beyond that the strategies one can used involve

- Tracking program memory use through screen watching and LOG file / process window contents examination.
- Tracking program memory use characteristics through profile and profiletol usage.
- Tracking memory use at a point through the DMPSYM option.
- Finding problems by searching and code isolation.

Each is described below.

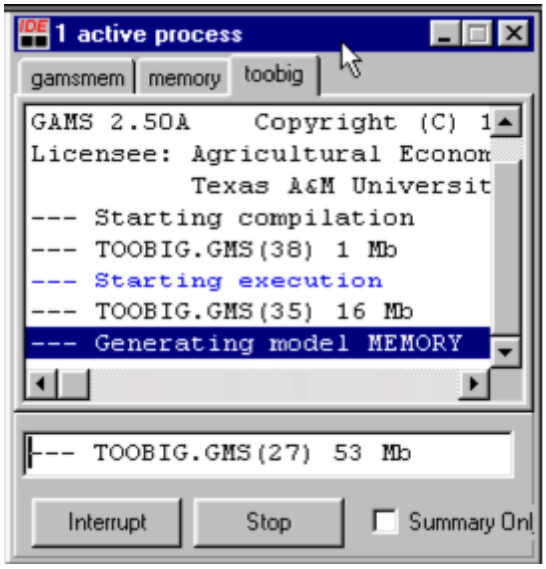
Screen watching and LOG file examination
Profile
Memory use dumps: Dmpsym
Looking within memory hogs to find offending term
My code won't work - searching

12.5.2.1 Screen watching and LOG file examination

GAMS outputs information on memory use to the screen during execution. Namely, the line number that is being executed and the current memory use are output. One may try to watch that display

© 2022 Prof. Bruce McCarl

although it can go by very fast. In particular, if can observe that the memory use jumps greatly during execution of a line number, then one would look at that line as a possible cause of excessive memory use. For example in the model in **toobig.gms** the procedure shows a large jump to 60+ megs in the reporting but the display goes by too fast to see just where. The LOG file and the IDE process window contain some memory information and the LOG file is often retained on the disk (this can be forced using the Logoption command line parameter) or can be opened by the IDE. But these items are not very useful because the output does not contain much detail.



Screen watching is not very satisfactory because

- Staring at the screen for long time periods may not be effective and one may miss certain statements, may identify statements improperly or get distracted and have to redo the approach repeatedly. Also statements may scroll by too quickly to observe.
- GAMS line reporting is misleading when loops and if statements are being executed. For example in running **gamsloop2.gms** the statements within the LOOP are all reported as if they were at the line number of the loop statement -28 in this case. Individual calculations in the loop are not reported to the screen i.e. lines 29-34. Thus through screen watching, one would not get any indication other than the loop is taking a lot of time and would not know where within the loop to look. The same thing happens within if statements and other GAMS control structures.

12.5.2.2 Profile

GAMS can give information on statement execution time and associated memory usage by employing the profile option. Profile is invoked using an option or a command line parameter, as I will discuss below. When GAMS is run with profile active the LST file contains profiling lines. These lines are collected and reproduced below for a run of the model **memory.gms**.

```
----      3              OTHER                 0.000      0.000 SECS     1.6 Mb
----      5              OTHER                 0.000      0.000 SECS     1.6 Mb
----      6              OTHER                 0.000      0.000 SECS     1.6 Mb
----     22 ASSIGNMENT y                       0.050      0.050 SECS     3.7 Mb
```

© 2022 Prof. Bruce McCarl

```
 78125
----     23 ASSIGNMENT x             0.060     0.110 SECS    8.4 Mb
 78125
----     24 ASSIGNMENT x             0.030     0.140 SECS    8.4 Mb
 78125
----     25 ASSIGNMENT q             0.000     0.140 SECS    8.4 Mb
 125
----     34           OTHER          0.000     0.140 SECS    8.4 Mb
generate
----     35 SOLVE INIT memory        0.000     0.140 SECS    8.4 Mb
----     29 EQUATION   z             0.460     0.600 SECS   27.8 Mb
 78125
----     31 EQUATION   res           0.000     0.600 SECS   27.8 Mb
 125
----     27 EQUATION   ob            0.341     0.941 SECS   29.9 Mb
   1
----     35 SOLVE FINI memory        0.340     1.281 SECS   29.9 Mb
----     35 GAMS FINI                1.032     2.313 SECS   29.9 Mb
----      1           EXEC-INIT      0.000     0.000 SECS   11.3 Mb
Solve
----     35 SOLVE READ memory        0.130     0.130 SECS   16.0 Mb
```

The profile contents by column are

- A demarking string ---- flagging this line. The line also contains the text string SECS (which is usually the better thing to search for).
- GAMS statement number of the instruction being profiled.
- The item name of the GAMS symbol being worked on.
- The execution time of each statement.
- Cumulative program execution time.
- Current memory use at end of statement.
- The number of cases for which the statement is executed.

12.5.2.2.1 Profiling to find memory hogging statements

Now lets look at what the profile is revealing about **memory.gms**. The inserted blue lines indicate execution stage and are not generated by GAMS.

```
----      3           OTHER          0.000     0.000 SECS    1.6 Mb
----      5           OTHER          0.000     0.000 SECS    1.6 Mb
----      6           OTHER          0.000     0.000 SECS    1.6 Mb
----     22 ASSIGNMENT y             0.050     0.050 SECS    3.7 Mb  78125
----     23 ASSIGNMENT x             0.060     0.110 SECS    8.4 Mb  78125
----     24 ASSIGNMENT x             0.030     0.140 SECS    8.4 Mb  78125
----     25 ASSIGNMENT q             0.000     0.140 SECS    8.4 Mb    125
----     34           OTHER          0.000     0.140 SECS    8.4 Mb
generate
----     35 SOLVE INIT memory        0.000     0.140 SECS    8.4 Mb
```

© 2022 Prof. Bruce McCarl

```
----     29 EQUATION   z              0.460     0.600 SECS   27.8 Mb  78125
----     31 EQUATION   res            0.000     0.600 SECS   27.8 Mb    125
----     27 EQUATION   ob             0.341     0.941 SECS   29.9 Mb      1
----     35 SOLVE FINI memory         0.340     1.281 SECS   29.9 Mb
----     35 GAMS FINI                 1.032     2.313 SECS   29.9 Mb
----      1           EXEC-INIT       0.000     0.000 SECS   11.3 Mb
Solve
----     35 SOLVE READ memory         0.130     0.130 SECS   16.0 Mb
```

Note the red lines show where memory use jumps are encountered (i.e., statements 22, 23 and 29. In turn, one can examine those statements to see if they can be reworked for memory use reduction.

12.5.2.2.1.1 Invoking profile

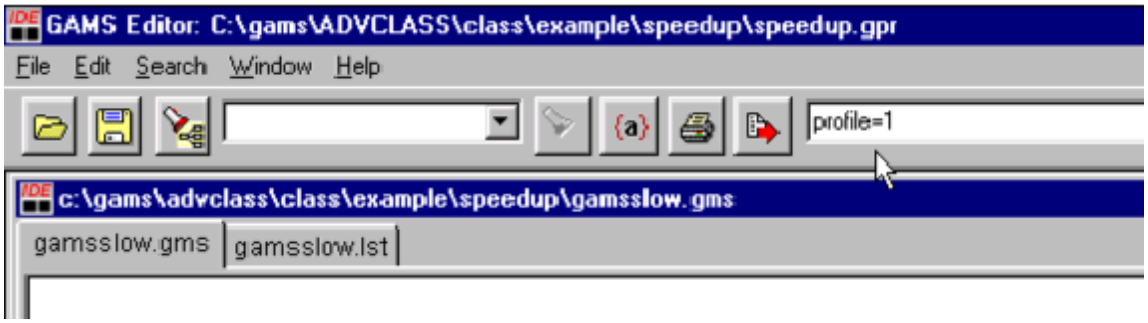
Profile may be invoked in a number of different ways.

On the GAMS command line
In the IDE GAMS parameters box
As an internal option

Profile is a valid command line option and may be set as follows

```
GAMS MYMODEL     PROFILE=1
```

One can enter the command line option in the IDE parameters box as shown below.



One can place an option statement into the program as follows:

```
OPTION PROFILE=3;
```

12.5.2.2.1.2 What should the profile number be

The profile statement is associated with a number i.e.

```
option profile=1    or    option profile=3
```

The use of higher numbers causes GAMS to carry out profiling within control structures such as loop, if,

© 2022 Prof. Bruce McCarl

repeat, for and when. The number tells how deep within the control structures to go.

If profile is set to

1 then GAMS reports statement timing and memory use at the control statement level without details on statements within control structures.

2 then GAMS includes output on statement timing and memory use for all statements that are not within control statements (everything but ifs or loops etc) plus the first level of statements within control structures.

3 then GAMS includes profile reports output on statement timing and memory use for all statements that are not within control statements and on statements nested within a second level of control statements (i.e. loops or ifs within loops).

gamsloop.gms provides an example. Note profile can have values of 4 and above.

12.5.2.2.1.3 Limiting profile output: Profiletol

Profile can generate a tremendous amount of output, much of which is not informative. The above profile reports show several statements reported for which there is not meaningful execution time. One can suppress this information by using a tolerance on the minimum amount of execution time in seconds that a statement must use to be reported.

```
OPTION PROFILETOL = 1;
```

or

```
OPTION PROFILETOL = 2;
```

or

```
OPTION PROFILETOL = 10;
```

This would limit reporting of statements to those that took 1 or 2 or 10 or more seconds of execution time. Bigger numbers yet can be used. Note, you must exercise care with use of profiletol during a memory use quest because there may be fast executing statements that use a lot of memory and a large value of profiletol could cause you to miss them.

12.5.2.3 Memory use dumps: Dmpsym

GAMS will report the number of cases stored for each item at the point in the program where the statement option dmpsym; is inserted. The cases stored report is in the LST file and for the example (**memdump.gms**) looks like

```
SYMBOL TABLE DUMP, NR ENTRIES = 62
ENTRY  ID          TYPE   DIM  DIM-OK  LENGTH DEFINED ASSIGNED DATAKNOWN

    1  MAPVAL      FUNCT    0   False       0   False   False     False
    2  CEIL        FUNCT    0   False       0   False   False     False
    3  FLOOR       FUNCT    0   False       0   False   False     False
    4  ROUND       FUNCT    0   False       0   False   False     False
    5  MOD         FUNCT    0   False       0   False   False     False
....
```

© 2022 Prof. Bruce McCarl

```
   40  TIMECLOSE    FUNCT   0   False       0   False   False     False
   41  ERRORLEVEL   FUNCT   0   False       0   False   False     False
   42  HEAPSIZE     FUNCT   0   False       0   False   False     False
   43  ***          FUNCT   0   False       0   False   False     False
   44  SAMEAS       PRED    0   False       0   False   False     False
   45  DIAG         PRED    0   False       0   False   False     False
   46  FILE         FILE    0   False       0   False   False     False
   47  I            SET     1    True       5    True   False      True
   48  J            SET     1    True       5    True   False      True
   49  K            SET     1    True       5    True   False      True
   50  L            SET     1    True       5    True   False      True
   51  M            SET     1    True       5    True   False      True
   52  N            SET     1    True       5    True   False      True
   53  O            SET     1    True       5    True   False      True
   54  Y            PARAM   7    True   78125   False    True     False
   55  Q            PARAM   3    True     125   False    True     False
   56  X            VAR     7    True   78125   False    True     False
   57  F            VAR     3    True     125   False    True     False
   58  OBJ          VAR     0    True       0   False    True     False
   59  Z            EQU     7    True   78125   False    True     False
   60  RES          EQU     3    True     125   False    True     False
   61  OB           EQU     0    True       0   False    True     False
   62  MEMORY       MODEL   0    True       3    True    True      True
```

The dump has 2 parts

- The first part contains information on GAMS functions that is of little user value.
- The second part contains a report of number of cases stored for each item in the GAMS program.

So let's focus on the second part for the example **memdump.gms**.

```
SYMBOL TABLE DUMP, NR ENTRIES = 62
ENTRY  ID           TYPE  DIM  DIM-OK  LENGTH DEFINED ASSIGNED DATAKNOWN

    1  MAPVAL       FUNCT   0   False       0   False   False     False
....
   46  FILE         FILE    0   False       0   False   False     False
   47  I            SET     1    True       5    True   False      True
   48  J            SET     1    True       5    True   False      True
   49  K            SET     1    True       5    True   False      True
   50  L            SET     1    True       5    True   False      True
   51  M            SET     1    True       5    True   False      True
   52  N            SET     1    True       5    True   False      True
   53  O            SET     1    True       5    True   False      True
   54  Y            PARAM   7    True   78125   False    True     False
   55  Q            PARAM   3    True     125   False    True     False
   56  X            VAR     7    True   78125   False    True     False
   57  F            VAR     3    True     125   False    True     False
```

© 2022 Prof. Bruce McCarl

```
58  OBJ       VAR     0    True        0    False    True     False
59  Z         EQU     7    True    78125    False    True     False
60  RES       EQU     3    True      125    False    True     False
61  OB        EQU     0    True        0    False    True     False
62  MEMORY    MODEL   0    True        3     True    True      True
```

In this dump, the columns of information that help in discovering large memory using items are

ID	User specified name for this item
TYPE	Item type (set, parameter, variable etc.)
DIM	Number of indices used in item definition
LENGTH	Number of cases (related to memory use)

In this display the rows where there are high counts in the length are associated with the items within the GAMS program which have large numbers of internal cases that must be stored which is associated with memory requirements. However, note that not all length counts are of equal significance. In particular, variables and equations use more memory per element than parameters that use more per element than sets (variables and equations have memory use for bounds, levels, marginals and scales for each case while parameter items are just one number per case and sets can be just one yes no indicator). In addition, note that set element explanatory text raises set element memory requirements.

Nevertheless one can look at the items (particularly variable, equations and parameters) that are of long length and make sure the number of cases stored are valid (reasons why it might not be are covered below).

Also note this dump gives status of active cases stored at the point in program where it is placed.

12.5.2.4 Looking within memory hogs to find offending term

Sometimes one runs profile and finds that the memory problem is inside a complex statement. For example, I have been known to regularly run models where report statements are well over 800 lines long containing 40 or so different terms. Finding a memory use problem in such a long statement still leaves one with the question of where is the problem. In such cases you can use the comment inserting syntax $Ontext/$Offtext and *'s to deactivate parts of the code.

Namely, if a multi-term piece of code spanning lines 1182 to 1294 uses excessive memory, one could would re-execute the code but split the term so say lines 1240 to 1294 are deactivated (surrounding them with an $Ontext - $Offtext sequence). Then if the remaining lines still showed use of a lot of memory, I would deactivate further code until the code deactivation greatly reduced memory use. I would then know that the last section put into the text status contained the memory hogging code. In turn, I could search further until the exact problematic portion is identified.

12.5.2.5 My code won't work - searching

Sometimes code leads to fatal computer errors which stops the job so you can't get any profile output (i.e. GAMS runs out of memory and dumps). But a memory overrun error causes the operating system buffer handing procedures to generally lose the last few lines of profile information when the job malfunctioned. So how do I find the problem? The answer is use

© 2022 Prof. Bruce McCarl

The problem reduction strategies in the small to large chapter

or

Search using code deactivation and isolation

The former involves the modeling simplification strategies discussed in the small to large chapter along with the profile and memory dump techniques discussed above and will not be elaborated on here. The latter merits discussion.

When one has code that just won't work one can engage in a quest for the last good statement. This can be done by using code deactivation by commenting out statements using

$Ontext/$Offtext to temporarily make comments of active statements
Making individual lines into comments using *

In turn, one would successively deactivate more and more statements until the performance changes with the job terminating appropriately. The searcher would then slowly activate members of the last code statements deactivated until the code performance got worse again. Then by iteratively activating and deactivating one can isolate the exact problematic terms. One may also use save restart to speed this up by saving the results up until a known good spot and just executing the suspect statements.

12.5.3 Causes of excessive memory use and repair

There are four big reasons a program may use too much memory. These are

Considering unnecessary cases
Keeping unnecessary items
Solving a huge problem
Inadequate computer capacity

I discuss the first two below and can only remark on the latter two that sometimes the problem must be simplified and sometimes one just needs to invest in bigger equipment. However, before doing either, consider using the techniques discussed above to insure the memory use is truly legitimate.

Also before beginning I wish to note that the material below is not only useful for the repair of specific models. Namely, by employing the memory reducing practices discussed herein in standard GAMS modeling, one can improve the effectiveness of any GAMS programs to which these techniques are applied. For example, the section on avoiding unnecessary cases will reveal conventions that should be used in all GAMS programming yielding memory use reductions without the need for memory usage investigations.

Avoiding considering unnecessary cases
Clearing memory of unnecessary items

© 2022 Prof. Bruce McCarl

12.5.3.1 Avoiding considering unnecessary cases

One way to reduce memory use in GAMS is to make sure only necessary cases are indexed. There are five contexts which merit consideration.

Calculation statements
Equation existence
Equation terms
Variable specifications
Memory traps to watch out for

Each is discussed below.

12.5.3.1.1 Calculation statements

A calculation of a parameter that is defined over a large of number of sets such as the following can inadvertently cover a huge number of cases

```
X(A, B, C, D, E) = 5
```

If each set had 20 members then the calculation would have to cover 3.2 million cases and would take a lot of memory. Narrowing attention only to good cases employing conditionals can help this

```
X(A, B, C, D, E) $ GOODCASE (A, B, C, D, E) = 5
```

where the user must somehow define the good case item, perhaps as a tuple.

12.5.3.1.2 Equation existence using conditionals

(transbad.gms vs transfix.gms)

Equations such as

```
slDEMANDEQ(MARKET)..
SUM(PLANT, SHIP(PLANT, MARKET))=G=DEMAND(MARKET);
```

may not really be relevant for all possible elements of the MARKET set and their uncontrolled generation involves GAMS needing to deal with more equations thus using more memory. I might only define equations when there is nonzero demand. Program execution memory use and model size would be reduced by

```
inDEMANDEQ(MARKET)$demand(market)..
SUM(PLANT,  SHIP(PLANT, MARKET)) =G= DEMAND(MARKET);
```

© 2022 Prof. Bruce McCarl

12.5.3.1.3 Equation term consideration limited using conditions

(transbad.gms vs transfix.gms)

An equation such as

```
slDEMANDEQ(MARKET)..
SUM(PLANT, SHIP(PLANT, MARKET))=G=DEMAND(MARKET);
```

can be defined and cause the problem to contain less variables and equations thus using less memory by adding conditionals both on equation and term existence to avoid considering unnecessary cases

```
fsDEMANDEQ(MARKET)$demand(market)..
  SUM(PLANT $cost(plant,market),SHIP(PLANT, MARKET))
    =G=
  DEMAND(MARKET);
```

12.5.3.1.4 Variable specification - suppression

(transbad.gms vs transfix.gms)

Cases can exist where unneeded variables are being defined. Creation of an unnecessary variable requires excess memory. For example, in a transport model I may only want variables that are defined over routes with nonzero transportation cost so I enter a conditional on cost.

```
TCOST =E= SUM((PLANT,MARKET)$cost(plant,market)
          , SHIPMENTS(PLANT,MARKET)*COST(PLANT,MARKET));
fsSUPPLYEQ(PLANT)$supply(plant)..
  SUM(MARKET $cost(plant,market), SHIPMENTS(PLANT, MARKET))=L=SUPPLY(PLANT);
fsDEMANDEQ(MARKET)$demand(market)..
  SUM(PLANT$cost(plant,market),  SHIPMENTS(PLANT, MARKET))=G=DEMAND(MARKET);
```

12.5.3.1.4.1 Watch out for incomplete suppression

(trnssu.gms)

When one uses conditionals to eliminate variables one must be careful to watch out for cases of incomplete elimination. Variables may still be there that you thought were gone. Below, I do not suppress zero transport cost cases in the bottom 2 equations and will end up with variables present that trivially allow demand satisfaction.

```
TCOST =E= SUM((PLANT,MARKET)$cost(plant,market)
                           , SHIPMENTS(PLANT,MARKET)*COST(PLANT,MARKET));
fsSUPPLYEQ(PLANT)$supply(plant)..
      SUM(MARKET, SHIPMENTS(PLANT, MARKET))=L=  SUPPLY(PLANT);
fsDEMANDEQ(MARKET)$demand(market)..
      SUM(PLANT,  SHIPMENTS(PLANT, MARKET)) =G= DEMAND(MARKET);
```

You need to suppress variables in a consistent fashion across all equations.

© 2022 Prof. Bruce McCarl

12.5.3.1.5 Memory traps to watch out for

There are a set of commands that user have been known to employ without appropriate caution that can use up a lot of memory. These generally involve variable attributes for scaling or upper/lower bounds. For example statements like

```
x.scale(i,j,k,l,m)=100;
x.up(i,j,k,l,m)=100;
x.lo(i,j,k,l,m)=100;
```

can use a lot of memory for irrelevant cases and one must again take care to restrict attention to relevant cases.

One can also import data from a database with long set explanatory text and cause problems.

12.5.3.2 Clearing memory of unnecessary items

In a GAMS run one can generate large temporary items which are not permanently required. For example one could have the sequence (**memtest.gms**)

```
Distance(I,j)=111;
Cost(I,j)=a+b*distance(I,j);
```

and never use distance again. One can cause GAMS to release the item from memory consumed using clear or killIn particular using option clear=itemname as follows

```
option clear=distance;
```

in effects zeros all entries in the distance matrix. The kill option also completely removes the item

```
option kill=distance;
```

In either case, the memory is not recovered unless the file is saved and restarted or a solve is executed with solvelink=0 .

Resetting a symbol to default values using "normal" assignment (i.e. x(i,j)=0;) is as fast as the execution time clear (or kill). The clear and kill are only more compact for for variables and equations where multiple statements would be needed to deal with each attribute (i.e. x.lo(i,j)=0;x.l(i,j)=0;x.up(i,j)=0 etc for scale, m and possibly others).

Compile time clean and kill, in the form of '$clear symbol' and '$kill symbol' also (re)set the symbol to default values but only have an effect on save and restart.

Use of $kill allows the used to redeclare the symbol with a data statement as follows:

```
set i /1,2,3/
$kill i
set i /a,b,c/
```

© 2022 Prof. Bruce McCarl

12.5.4 Limiting memory use with HeapLimit

Cases may arise where one needs to limit the amount of memory a GAMS job can use. The HeapLimit GAMS parameter and function permits this to be done during GAMS compilation and execution, but does not limit memory use during solver execution (i.e. not during CONOPT or CPLEX etc.). It limits memory use to a given number of MegaBytes. If the data storage exceeds this limit, the job will be terminated with return code 10, out of memory. These features may be especially useful in a server environment. This is accomplished in one of two ways. Namely through

- The GAMS parameter `HeapLimit` sets the limit of memory use at compile and execution time for a GAMS job.
- The function/property `HeapLimit` can be used to interrogate the current limit and allows it to be reset.

Note the NLP solver CONOPT also has a `HeapLimit` option which ensures that the solver will not use more dynamic memory than allowed.

13 More Language Features

This section covers slightly more advanced features of the GAMS language. The coverage is organized by chapter with the chapters covering:

Including External Files
Dollar Commands
The Option Command

13.1 Including External Files

GAMS may include external files. This may be done with and without substitution of some items within the file. There are also special provisions regarding inclusion of comma-delimited files.

Inclusion without arguments
Suppressing the listing of include files
Redefining the location of include files - Idir
Include with arguments
Influence on LST file contents: $Oninclude and $Offinclude
Passing $ commands between code segments: $Onglobal and $Offglobal
Special provision for CSV files

13.1.1 Inclusion without arguments

When files of GAMS instructions or data are to be incorporated into a GAMS program and one simply

© 2022 Prof. Bruce McCarl

wants to incorporate the file as is one uses the GAMS dollar command $Include. Otherwise one may wish to specify some arguments and use the include with arguments commands.

$Include

13.1.1.1 $Include

One may include external files using the syntax

```
$Include externalfilename
```

where the externalfilename can

- Include the full path or just a file name relative to the current working directory (which will be the IDE project file location) using no path name or syntax like ./subdir/externalfilename that goes into a subdirectory below this one or a .. like ../filename which goes above this directory.
- Be quoted or unquoted.
- Only contain part of name and does not need to incorporate the extension gms. In turn if
  - A file with just the name given in the $Include is found in the current working directory (which will be the IDE project file location), then it will be used.
  - If not a file with the name externalfilename.gms will be searched for in the current working directory (which will be the IDE project file location) and if found, then it will be used.

Example:

Suppose we break a transport problem into three files (**trandata.gms**, **tranmodl.gms**, **tranrept.gms**) and include them into one composite file **tranint.gms** as follows

```
$Include trandata
$Include tranmodl
$Include tranrept
```

Note also **tranrept.gms** includes another file **trannest.gms**. The resultant LST file looks as follows

```
INCLUDE    D:\GAMSPDF\TRANDATA.GMS
    2  SETS  PLANT    PLANT LOCATIONS  /NEWYORK  , CHICAGO , LOSANGLS /
    3        MARKET   DEMAND MARKETS   /MIAMI,   HOUSTON, MINEPLIS, PORTLAND/
    4  PARAMETERS   SUPPLY(PLANT)  QUANTITY AVAILABLE AT EACH PLANT
    5                  /NEWYORK   100, CHICAGO   275, LOSANGLS   90/
    6               DEMAND(MARKET)   QUANTITY REQUIRED BY DEMAND MARKET
    7                  /MIAMI 100,HOUSTON 90,MINEPLIS 120,PORTLAND 90/;
    8   TABLE   DISTANCE(PLANT,MARKET)   DISTANCE FROM EACH PLANT TO EACH MARKE
    9                        MIAMI   HOUSTON   MINEPLIS   PORTLAND
   10           NEWYORK      1300     1800       1100       3600
   11           CHICAGO      2200     1300        700       2900
   12           LOSANGLS     3700     2400       2500       1100      ;
```

© 2022 Prof. Bruce McCarl

```
INCLUDE    D:\GAMSPDF\TRANMODL.GMS
   14  PARAMETER COST(PLANT,MARKET)    CALCULATED COST OF MOVING GOODS;
   15            COST(PLANT,MARKET) = 50 + 1 * DISTANCE(PLANT,MARKET);
   16   POSITIVE VARIABLES
   17          SHIPMENTS(PLANT,MARKET) AMOUNT SHIPPED OVER A TRANSPORT ROUTE;
   18   VARIABLES TCOST                TOTAL COST OF SHIPPING OVER ALL ROUTES;
   19   EQUATIONS TCOSTEQ              TOTAL COST ACCOUNTING EQUATION
   20             SUPPLYEQ(PLANT)      LIMIT ON SUPPLY AVAILABLE AT A PLANT
   21             DEMANDEQ(MARKET)     MINIMUM REQUIREMENT AT A DEMAND MARKET;
   22   TCOSTEQ.. TCOST =E=SUM((PLANT,MARKET), SHIPMENTS(PLANT,MARKET)*
   23                                                  COST(PLANT,MARKET));
   24   SUPPLYEQ(PLANT).. SUM(MARKET,SHIPMENTS(PLANT,MARKET))=L=SUPPLY(PLANT);
   25   DEMANDEQ(MARKET)..SUM(PLANT,SHIPMENTS(PLANT,MARKET))=G=DEMAND(MARKET);
   26   MODEL TRANSPORT /ALL/;
   27   SOLVE TRANSPORT USING LP MINIMIZING TCOST;
INCLUDE    D:\GAMSPDF\TRANREPT.GMS
   29  PARAMETER MOVEMENT(*,*)  COMMODITY MOVEMENT;
   30  MOVEMENT(PLANT,MARKET)=SHIPMENTS.L(PLANT,MARKET);
   31  MOVEMENT("TOTAL",MARKET)=SUM(PLANT,SHIPMENTS.L(PLANT,MARKET));
   32  MOVEMENT(PLANT,"TOTAL")=SUM(MARKET,SHIPMENTS.L(PLANT,MARKET));
   33  MOVEMENT("TOTAL","TOTAL")=SUM(MARKET,MOVEMENT("TOTAL",MARKET));
   34  OPTION DECIMALS=0;
   35  DISPLAY MOVEMENT;
   36  PARAMETER COSTS(*,*)  COMMODITY MOVEMENT COSTS BY ROUTE;
   37  COSTS(PLANT,MARKET)=COST(PLANT,MARKET)*SHIPMENTS.L(PLANT,MARKET);
   38  COSTS("TOTAL",MARKET)
   39      =SUM(PLANT,COST(PLANT,MARKET)*SHIPMENTS.L(PLANT,MARKET));
   40  COSTS(PLANT,"TOTAL")
   41      =SUM(MARKET,COST(PLANT,MARKET)*SHIPMENTS.L(PLANT,MARKET));
   42  COSTS("TOTAL","TOTAL")=TCOST.L;
   43  OPTION DECIMALS=0;
   44  DISPLAY COSTS;
   45  PARAMETER SUPPLYREP(PLANT,*)  SUPPLY REPORT;
   46  SUPPLYREP(PLANT,"AVAILABLE")=SUPPLY(PLANT);
   47  SUPPLYREP(PLANT,"USED")=MOVEMENT(PLANT,"TOTAL");
   48  SUPPLYREP(PLANT,"MARGVALUE")=ABS(SUPPLYEQ.M(PLANT));
   49  OPTION DECIMALS=2;
   50  DISPLAY SUPPLYREP;
   51  PARAMETER DEMANDREP(MARKET,*)  DEMAND REPORT;
   52  DEMANDREP(MARKET,"REQUIRED")=DEMAND(MARKET);
   53  DEMANDREP(MARKET,"RECIEVED")=MOVEMENT("TOTAL",MARKET);
   54  DEMANDREP(MARKET,"MARGCOST")=ABS(DEMANDEQ.M(MARKET));
   55  OPTION DECIMALS=2;
   56  DISPLAY DEMANDREP;
   57  PARAMETER CMOVEMENT(*,*) COSTS OF CHANGING COMMODITY MOVEMENT PATTERN;
   58  CMOVEMENT(PLANT,MARKET)=SHIPMENTS.M(PLANT,MARKET);
   59  OPTION DECIMALS=2;
   60  DISPLAY CMOVEMENT;
INCLUDE    D:\GAMSPDF\TRANNEST.GMS
   61 *this is a nested include
```

© 2022 Prof. Bruce McCarl

and runs just as if it were one continuously typed file.

Notes:

- In the LST file the incidence of the include file is marked in two ways.
  - Wherever an include file starts the echo print contains a line that indicates the file name and location. In the echo print of the above example 4 such lines appear as shown just below. Note as illustrated in the LST file above these are separated by a echo print of the content of the included files and are just grouped together here for convenience of exposition.

```
INCLUDE    C:\GAMS\ADVCLASS\CLASS\EXAMPLE\LINK\TRANDATA.GMS
INCLUDE    C:\GAMS\ADVCLASS\CLASS\EXAMPLE\LINK\TRANMODL.GMS
INCLUDE    C:\GAMS\ADVCLASS\CLASS\EXAMPLE\LINK\TRANREPT.GMS
INCLUDE    D:\GAMSPDF\TRANNEST.GMS
```

  - At the end of the file one gets an indication of what was inserted and where

```
     SEQ   GLOBAL TYPE      PARENT          LOCAL FILENAME

1    1     INPUT            0       0       C:\GAMSPDF\TRANINT.G
2    6     INCLUDE          1       6       .C:\GAMSPDF\TRANDATA
3    25    INCLUDE          1       7       .C:\GAMSPDF\TRANMODL
4    44    INCLUDE          1       8       .C:\GAMSPDF\TRANREPT
5    77    INCLUDE          4       33      ..C:\GAMSPDF\TRANNES
```

There are several columns to this display

- The **SEQ** column gives a number to the include files encountered always including the base GMS file which is listed as INPUT (**tranint.gms**).
- The **GLOBAL** column gives the line number within the composite LST file (which is expanded for the presence of the included files as above) where the $Include statement occurs.
- The **TYPE** column indicates the type of include present. The various types can be INPUT, INCLUDE, BATINCLUDE, LIBINCLUDE, and SYSINCLUDE. INPUT is used to label the base file (**tranint.gms**). The other modifiers in front of the word include will be discussed below.
- The **PARENT** column provides the number of the file from the SEQ column into which this file was included (note in the above example most were included in file number 1 the base **Input** but that **trannest.gms** was included in file number 4 which is **tranrept.gms**).
- The **LOCAL** column gives the local line number in the parent file where the $Include appeared (showing the first three were included in lines 6, 7 and 8 of the INPUT file **tranint.gms** and that **trannest.gms** is included into line 33 of **tranrept.gms**.
- The **FILENAME** column gives the path and name for the included file.

13.1.1.1.1 Includes that cause compiler error messages

When syntax errors are made in the referenced files then the LST file has additional information about the name of the include file and the local line number (**incerr.gms**) as follows

```
INCLUDE     D:\GAMSPDF\TRANMODL.GMS
```

© 2022 Prof. Bruce McCarl

```
   15  PARAMETER COST(PLANT,MARKET)    CALCULATED COST OF MOVING GOODS;
   16            COST(PLANT,MARKET) = 50 + 1 * DISTANCE(PLANT,MARKET);
   17   POSITIVE VARIABLES
   18          SHIPMENTS(PLANT,MARKET) AMOUNT SHIPPED OVER A TRANSPORT ROUTE;
   19   VARIABLES TCOST                 TOTAL COST OF SHIPPING OVER ALL ROUTES;
   20   EQUATIONS TCOSTEQ               TOTAL COST ACCOUNTING EQUATION
   21             SUPPLYEQ(PLANT)       LIMIT ON SUPPLY AVAILABLE AT A PLANT
   22             DEMANDEQ(MARKET)      MINIMUM REQUIREMENT AT A DEMAND MARKET;
****                    $195
**** LINE      8 IN FILE D:\GAMSPDF\TRANMODL.GMS
```

Note here the error message indicates the error is not in line 22 of the expanded listing but rather in line 8 of the included file. The IDE also jumps to the relevant line in the included file when the error discovery procedure is employed.

13.1.2 Suppressing the listing of include files

Sometimes the files included are large files that one really does not wish to be included in the echo print within the LST file. Under such circumstances one can do two things.

- One can suppress all include file listings (and the alternative Batinclude etc forms below) using the $offlisting syntax.
- One can use a $offlisting somewhere within the included files (say after the first four lines just to give a small taste of the contents and all subsequent lines in the files will be suppressed. GAMS then automatically with switch back to an $onlisting status in dealing with the subsequent files.

13.1.3 Redefining the location of include files - Idir

The directory in which $Include files are expected to be located can be altered. This is done by using the IDIR command line parameter in which case the named file is looked for first then one with a .gms extension.

13.1.4 Include with arguments

There are variants of the include command which permit insertion of some user defined arguments in the file to be included. Three of these variants exist Batinclude, Libinclude and Sysinclude.

$Batinclude
$Libinclude
$Sysinclude

13.1.4.1 $Batinclude

There are 2 main attributes to Batinclude that differentiate it from the simple Include discussed above and the Libinclude and Sysinclude forms discussed below. These involve the way that parameter inclusion works and location of the file to be included. The Batinclude files are located in the current working directory or in the search path identified by the Idir command line parameter.

© 2022 Prof. Bruce McCarl

The basic syntax for the command is

```
$Batinclude externalfilename argument1 argument2 …
```

13.1.4.1.1 How parameter inclusion works

Batinclude passes user defined arguments into selected places within the included file. Argument1, argument2,.. are arguments passed for substitution. These arguments can be single unbroken strings (quoted or unquoted) or quoted multi-part strings that include spaces and special characters.

The arguments are treated as character strings that are substituted by argument number inside the included file as in the DOS batch facility. Argument substitution is indicated by using the character % followed immediately by the argument number where %1 refers to the first argument, %2 to the second argument, and so on.

Names may be used for the substitutable parameters using the procedure discussed under the $setargs command.

Example:

Suppose we wish to do addition involving different elements in fact suppose we have scalars a and b with data and wish to form

```
c = a + b;
d = a + c;
```

We can accomplish this with a Batinclude file. In particular, we can write a file that sets

```
Argument1 = argument2 + argument3
```

and call it twice

```
once with arguments c , a , b
again with arguments d , a , c
```

To do this we build a file that contains

```
%1 = %2 + %3 ;
```

and call it **batincadd.gms**.

Then we build **addbat.gms** which will include the above file twice using the syntax

```
scalar a /2/, b /4/, c ,d  ;
$Batinclude batincadd c a b
$Batinclude batincadd d a c
```

In turn the LST file from the run shows the following

```
  1  *main program for Batinclude example
  2  scalar a /2/, b /4/, c ,d  ;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\BATINCADD.GMS
```

© 2022 Prof. Bruce McCarl

```
   4  *example of arguments in Batinclude
   5  c = a + b;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\BATINCADD.GMS
   7  *example of arguments in Batinclude
   8  d = a + c;
```

Note in line 2 of the source the statement

```
$Batinclude batincadd c a b
```

has three arguments (c a b) attached to the call of the included file **batincadd.gms** and that this causes a substitution of the parameters into

```
%1 = %2 + %3 ;
```

where c is the first argument and goes into place %1
a is the second argument and goes into place **%2**
b is the third argument and goes into place **%3**

yielding

```
c = a + b;
```

in line 3 of the LST file.

The equivalent example using setargs is in **setargs.gms** and **batincsag.gms** which causes the Batinclude file to use number1 in place of %1, second in place of %2 , house in place of %3 and * for all remaining using code like

```
$setargs number1 second house *
%number1% = %second% * %house%;
```

in the BATINCLUDE instead of

```
%1=%2*%3;
```

Note setargs must appear in the Batincluded file. Also note that the * or a . or a / will cause a numbered item to be skipped over.

Notes:

- Arguments are substituted as text items.
- All substitutions are done before the compiler is activated.
- Arguments are not substituted in comments.
- GAMS requires that the line after substitutions cannot be longer than the maximum input line length (currently 32767 characters).
- The font upper and lower casing of the passed parameters is preserved for use in string comparisons.
- If the argument number does not correspond to a non blank argument, then a null string is substituted.

© 2022 Prof. Bruce McCarl

- When the argument %0 is used then the full file and file path name for the current file will be included.
- A $Batinclude call without any arguments is equivalent to a $Include call.
- Includes with parameters are useful when wishes to do the same general operation over a number of named parameters or files.
- A $Shift command can be used in argument processing.

Additional Examples:

The full power of a include with arguments can best be appreciated by expanding the example

Adding the line below to **addbat.gms**

```
$Batinclude batincadd d "a+b-d*a" "c*sqrt(abs(a+1))"
```

results in an LST file with

```
10  d = a+b-d*a c*sqrt(abs(a+1));
```

showing that quoted items for arguments can result in substitution of formulae in the included file.

Adding a fourth argument to activation of the Batinclude file (**addbat.gms**) in a slightly different file **batincadd2.gms**

```
$Batinclude batincadd2 d a c "text for the display"
```

and adding the line to the included file

```
display "%4", "%0"
```

yields the LST file component

```
display "text for the display", "C:\GAMS\GAMSPDF\BIGONE\BATINCADD2.GMS"
```

and the display output

```
----     16 text for the display
            D:\GAMSPDF\BATINCADD2.GMS
```

More complex examples also appear in the conditional compilation chapter.

Note the parameter %0 gives the name of the file being incorporated by the Batinclude.

13.1.4.2 $Libinclude

One can also pass parameters using the Libinclude syntax that only really 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.

The basic syntax for the command is

```
$Libinclude externalfilename argument1 argument2 …
```

© 2022 Prof. Bruce McCarl

Notes:

- By default, the library include directory is set to the inclib subdirectory of the GAMS system directory.
- On a PC running version 22.7 in the default library include directory location this would be

```
c:\program files\gams22.7\inclib\
```

- This is useful when one develops a general utility that is to be used in many settings across many directories.

13.1.4.2.1 Ldir

The default directory can be reset with the Ldir command line parameter using the syntax

```
GAMS myfile LDIR=c:\whereiwant
```

13.1.4.3 $Sysinclude

One can also pass parameters using the Sysinclude syntax that only really differs from Batinclude in where the file comes from. Namely, if an incomplete path is given, the file name is completed using the system include directory.

The basic syntax for the command is

```
$Sysinclude externalfilename argument1 argument2 …
```

Notes:

- By default, the system include directory is set to the GAMS system directory.
- On a PC running version 22.7 the default system include directory location this would be

```
c:\program files\gams22.7\
```

- This is useful when one develops a general utility that is to be used in many settings across many directories.

13.1.4.3.1 Sdir

The default directory can be reset with the Sdir command line parameter using the syntax

```
GAMS myfile SDIR=c:\whereiwant
```

13.1.5 Influence on LST file contents: $Oninclude and $Offinclude

Ordinarily the LST file contains an echo print of the contents of all included files with contents expanded for any passed arguments. On can exclude such listings using

- a $Offinclude command which can be subsequently reversed using $Oninclude.

© 2022 Prof. Bruce McCarl

- a $Offlisting in a file to be include which does not need to be matched by a $Onlisting as GAMS will automatically reset the echo print to on text status after the file is included.

13.1.6 Passing $ commands between code segments: $Onglobal and $Offglobal

Ordinarily the $commands in a program which includes others are not passed on to included files. On can require GAMS to pass such commands using the $Onglobal command which can be subsequently reversed using $Offglobal.

13.1.7 Special provision for CSV files

Spreadsheets and other programs can read and write CSV (comma separated value) files. Therein commas separate fields and text items can be in quotes. GAMS can also include such files using the $ command $Ondelim.

$Ondelim$Offdelim

13.1.7.1 $Ondelim and $Offdelim

CSV formatted entries files may be incorporated by using the command $Ondelim before beginning the entry and then $Offdelim afterwards. Consider the following example (**CSV1.gms**):

```
SETS
   PLANT  PLANT LOCATIONS /NEWYORK,CHICAGO,LOSANGLS /
   MARKET DEMANDS  /MIAMI,HOUSTON, PORTLAND/
table datafrmCSV(plant,market)  data in CSV format
$Ondelim
dummy,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGLS,3700,2400,2500
$Offdelim
```

where the section in blue is in CSV form. In version 22.7 and later one can also drop the dummy entry but must maintain the comma as follows

```
table datafrmCSV2(plant,market)  data in CSV format
$Ondelim
,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGLS,3700,2400,2500
$Offdelim
```

One could equivalently use an include file as follows (**CSV2.gms**)

```
SETS
   PLANT  PLANT LOCATIONS /NEWYORK,CHICAGO,LOSANGLS /
```

© 2022 Prof. Bruce McCarl

```
   MARKET DEMANDS  /MIAMI,HOUSTON, PORTLAND/
table datafrmCSV(plant,market)  data in CSV format
$Ondelim
$Include Csvtoinc
$Offdelim
```

where the file **csvtoinc.gms** contains

```
dummy,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGLS,3700,2400,2500
```

where the dummy could be dropped but the comma must be maintained.

Notes:

- We use $Ondelim before the CSV entry or included file and $Offdelim after to tell GAMS that the following entries are delimited.
- A text item like the entry dummy above is included to use up the space over the set elements defining the table rows although as of version 22.7 this can be balk with just a comma.

13.2 Dollar Commands

Dollar commands set off with a $ in column 1 OR A $$ in other columns are used to exercise increased control over GAMS functions. All $ commands in this class are implemented at compile time and cannot be data value dependent. (Note we are not dealing here with $ in conditionals.)

Basics
Categories of $ commands
Detailed description of dollar commands

13.2.1 Basics

Dollar commands must start with a "$" symbol in the first column or with a $$ in columns other than one. Dollar commands may be placed anywhere within a GAMS program and are processed during the compilation of the program. The commands options are used as follows

```
$commandname argument
```

or

```
$$commandname argument
```

where

$ begins command in column one

$$ begins command in columns other than one

commandname is the name of the specific option to be altered as listed below
Argument is a potentially needed argument associated with the command

Notes:

- Dollar commands ordinarily occur in column 1
- .Dollar commands can occur in columns other than 1 if they begin with $$
- .Dollar commands do not appear on the compiler listing unless an error has been detected or they are included using the option $Ondollar.
- The command names are not case sensitive.
- Depending on the particular dollar command, the number of arguments required can vary from 0 to many.
- Blank space is permitted between the $ character and the command name that follows.
- In most cases, multiple dollar commands can be processed on a line although $$ can only occur once.
- The effect of the dollar command is felt immediately after the dollar command is processed.
- Dollar commands ordinarily begin with a $ but this can be changed using $dollar.
- Dollar commands are processed at compile time immediately with the program stopping until the command is finished and all $ commands are resolved before beginning execution. This can lead to interesting implications for the results as discussed next.

13.2.1.1 When do dollar commands occur?

One important consideration when considering employing $commands involves the timing of there resolution. Suppose we have a file (**toinclude.gms**) that contains

```
a=3;
```

and a file that includes this file but also redefines it via use of put commands and in the sequence as typed the redefinition occurs before the inclusion as follows (**timeinclude.gms**)

```
scalar a /1/;
file toinc /toinclude.gms/;
put toinc,'a=5;' /;
$include toinclude
display a;
```

so that after the program runs the file (**toinclude.gms**) contains

```
a=5;
```

© 2022 Prof. Bruce McCarl

So what does the display look like? Well it is

```
6 PARAMETER a                    =        3.000
```

reflecting the initial value of the parameter "a" from **toinclude.gms** before the program started, and is unaffected by the action of the put command. Why? This occurs because all $ commands are resolved at compile time and can never be affected by the subsequent results of the program.

Another example is also instructive as illustrated by the example **dollartime.gms** below.

```
1  set i /i1,i2/
2  $ONMULTI
3  parameter a(i) /i1 22, i2 33/;
4  display a;
5  parameter a/i1 44/;
6  display a;
```

where the resultant output is

```
----      4 PARAMETER a
i1 44.000,    i2 33.000

----      6 PARAMETER a
i1 44.000,    i2 33.000
```

Note in line 4 the value we get from the display is not the 22 entered originally for a(1) but is rather the redefined value allowed by $ONMULTI of 44 which does not actually occur until line 5. This occurs because the $ and redefinition is processed before the execution time display occurs.

This can become yet more complex when one is using $CALL and GDX associated commands as illustrated in the Links to Other Programs Including Spreadsheets chapter.

13.2.2 Categories of $ commands

The Dollar Commands can be grouped into major functional categories affecting

$$ for commands starting beyond column 1
Commands for inclusion of comments
LST and other output file contents
Inclusion of external files
Contents dependent compilation
Numerical procedures used
Reset of data for items
GDX files
Compiler procedures
Execution of an external program
Impose data access restrictions

© 2022 Prof. Bruce McCarl

Tear apart strings
Compress and encrypt files
Macro definitions
Include user defined function

Here we list these commands by group then later we define them more precisely.

13.2.2.1 Commands for inclusion of comments

The dollar commands in this group cause GAMS to include comments or allow different forms of comments. They are discussed here and in the Comment chapter. These dollar commands are:

$Comment	Dollar command that changes character used to start a comment in column 1 which is now an *.
$Eolcom	Dollar command that changes delimiter for end of line comments.
$Hidden	Dollar command that inserts one line comment that does not appear in LST file.
$Inlinecom	Dollar command that changes character strings delimiting an in line comment.
$Maxcol	Dollar command that sets right margin for the input file.
$Mincol	Dollar command that sets left margin for the input file.
$Offeolcom	Dollar command that deactivates ability to use end-of-line comments.
$Offinline	Dollar command that deactivates ability to use in line comments.
$Offmargin	Dollar command that turns off mincol and maxcol margins.
$Offnestcom	Dollar command that prohibits nested in line comments.
$Offtext	Dollar command that ends a multi line comment.
$Oneolcom	Dollar command that activates ability to use end-of-line comments.
$Oninline	Dollar command that activates ability to use in line comments.
$Onmargin	Dollar command that turns on mincol and maxcol margins.
$Onnestcom	Dollar command that allows nested in line comments.
$Ontext	Dollar command that starts a multi line comment.
$Remark	Dollar command that includes a comment with a substitutable parameter.

13.2.2.2 LST and other output file contents control

The dollar commands in this group cause GAMS to alter the contents of the LST file or send messages to other files. They are discussed here and in the Standard Output chapter. The dollar commands are:

$Abort	Dollar command that causes compilation to stop and issues an error message in LST file.
$Double	Dollar command that starts double spacing of echo print lines in LST file.
$Echo	Dollar command that echoes text to a named file.
$Eject	Dollar command that starts a new page in LST file.
$Error	Dollar command that causes reporting of compiler error to LST file but allows continued compilation.
$Lines	Dollar command that starts new page if less than n lines are left on a page.
$Log	Dollar command that sends specified text to the LOG file.
$Offdollar	Dollar command that suppresses echo print of dollar commands to LST file.
$Offecho	Dollar command to stop action of $Onecho, $Onechos or $Onechov.
$Offinclude	Dollar command that suppresses echo print of included files to LST file.
$Offlisting	Dollar command that deactivates echo print of subsequent input lines.
$Offlog	Dollar command that turns off line logging.

© 2022 Prof. Bruce McCarl

$Offput	Dollar command that stops $onput transfer of text to a put file.
$Offsymlist	Dollar command that removes symbol list from LST file.
$Offsymxref	Dollar command that removes symbol cross reference from LST file.
$Offuellist	Dollar command that removes unique element list from LST file.
$Offuelxref	Dollar command that removes unique element cross reference from LST file.
$Offverbatim	Dollar command used in conjunction with the GAMS parameter DUMPOPT
$Ondollar	Dollar command that adds echo print of dollar commands to LST file.
$Onecho	Dollar command to start copying succeeding lines to file.
$Onechos	Dollar command to start copying succeeding lines to file with parameter substitution.
$Onechov	Dollar command to start copying succeeding lines to file.
$Oninclude	Dollar command that causes echo print of included files.
$Onlisting	Dollar command that activates echo print of subsequent input lines.
$Onlog	Dollar command that resets line logging.
$Onput	Dollar command that transfers the text in subsequent lines to a put file.
$Onputs	Dollar command that transfers the text in subsequent lines to a put file with parameter substitution.
$Onputv	Dollar command that transfers the text in subsequent lines to a put file without parameter substitution.
$Onsymlist	Dollar command that includes symbol list in LST file.
$Onsymxref	Dollar command that includes symbol cross reference in LST file.
$Onuellist	Dollar command that adds unique element list to LST file.
$Onuelxref	Dollar command that adds unique element cross reference to LST file.
$Onverbatim	Dollar command used in conjunction with the GAMS parameter DUMPOPT
$Single	Dollar command that starts single space listing of subsequent echo print lines in LST file.
$Stars	Dollar command that redefines characters for four **** messages.
$Stitle	Dollar command that defines subtitle for LST file.
$Title	Dollar command that defines LST file title.

13.2.2.3 Ways of including external files

The dollar commands in this group cause GAMS to include external files in a program and control attributes of those files plus the LST file. They are discussed here and in the file inclusion chapter. The dollar commands are:

$Batinclude	Dollar command that includes an external file with arguments.
$Include	Dollar command that includes an external file without arguments.
$Libinclude	Dollar command that includes a file with arguments from inclib subdirectory.
$Offglobal	Dollar command that causes dollar commands in main programs to not be honored in included files.
$Offinclude	Dollar command that suppresses echo print of included files in LST file.
$Offrecurse	Dollar command that disallows a file to include itself.
$Onglobal	Dollar command that causes dollar commands in main programs to be honored in included files.
$Oninclude	Dollar command that causes echo print of included files.
$Onrecurse	Dollar command that allows a file to include itself.
$Shift	Dollar command that shifts arguments in include files.
$Sysinclude	Dollar command that includes file with arguments from system directory.

13.2.2.4 Contents dependent compilation

The dollar commands in this group cause GAMS to execute statements that follow different procedures

© 2022 Prof. Bruce McCarl

based on the setting of flags or the characteristics of data items or file existence. They are discussed briefly here and more extensively in the Conditional Compilation chapter. The dollar commands are.

$Abort	Dollar command that causes compilation to stop and issues an error message in LST file.
$Drop	Dollar command that destroys a variable that was defined with $Set.
$Dropenv	Dollar command that destroys a variable that was defined with $Setenv.
$Dropglobal	Dollar command that destroys a variable that was defined with $Setglobal.
$Droplocal	Dollar command that destroys a variable that was defined with $Setlocal.
$Else	Dollar command that is followed by an instruction which is executed if the matching $ifThen statement is not true
$Elseif	Dollar command that controls whether a number of statements are active
$Elseife	Dollar command that controls whether a number of statements are active (expression evaluation)
$Elseifi	Dollar command that controls whether a number of statements are active (case insensitive)
$Endif	Dollar command that hast to match with $Ifthen, $Ifthene or $Iftheni
$Error	Dollar command that causes reporting of compiler error to LST file but allows continued compilation.
$Eval	Dollar command that evaluates a numerical expression and places it into a scoped control variable.
$Evalglobal	Dollar command that evaluates a numerical expression and places it into a global control variable.
$Evallocal	Dollar command that evaluates a numerical expression and places it into a local control variable.
$Goto	Dollar command that transfers control to a line with an internal label.
$If	Dollar command that causes a statement to be executed at compile time if case sensitive conditional is true.
$If not	Dollar command that causes a statement to be executed at compile time if case sensitive conditional is false.
$Ife	Dollar command that causes a statement to be executed at compile time if evaluated conditional is true.
$Ife not	Dollar command that causes a statement to be executed at compile time if evaluated conditional is false.
$Ifi	Dollar command that causes a statement to be executed at compile time if case sensitive conditional is true.
$Ifi not	Dollar command that causes a statement to be executed at compile time if case sensitive conditional is false.
$Ifthen	Dollar command that controls whether a number of statements are active
$Ifthene	Dollar command that controls whether a number of statements are active (expression evaluation)
$Iftheni	Dollar command that controls whether a number of statements are active (case insensitive)
$Label	Labels a line allowing branching to it from a $goto.
$Maxgoto	Maximum number of jumps to the same label.
$Prefixpath	Augments search path windows environment variable.
$Set	Dollar command that defines control variable.
$Setenv	Dollar command that defines or redefines windows environment variable.
$Setglobal	Dollar command that defines global control variable.
$Setlocal	Dollar command that defines local control variable.
$Terminate	Dollar command that terminates compilation and execution immediately.

© 2022 Prof. Bruce McCarl

$Warning	Dollar command that issues a compilation warning but continues compilation and execution.

13.2.2.5 Alter numerical procedures used

The dollar commands in this group alter some of the GAMS numeric procedures. The dollar commands are.

$Offdigit	Dollar command that deactivates significant digit transformation.
$Offeps	Dollar command that deactivates treatment of zeros as EPS.
$Ondigit	Dollar command that activates significant digit transformation.
$Oneps	Dollar command that activates treatment of zeros as EPS.

13.2.2.6 Alter data for items

The dollar commands in this group allow removal of data items or a reset of their contents. The dollar commands in this group are listed in the table below.

$Clear	Dollar command that resets named items to all zero values.
$Kill	Dollar command that removes all data for an item and the item itself allowing redefinition of items.

13.2.2.7 GDX file read/write

The dollar commands in this group allow one to pass data to and from GDX files as discussed in the GDX chapter. The dollar commands in this group are:

$Gdxin	Dollar command that opens/closes a GDX file for input.
$Gdxout	Dollar command that opens/closes a GDX file for output.
$Load	Dollar command that loads data from a GDX file.
$Loaddc	Dollar command that loads data from a GDX file with domain checking.
$Loaddcm	Dollar command that loads data from a GDX file (mixture of $Loaddc and $Loadm).
$Loaddcr	Dollar command that loads data from a GDX file (mixture of $Loaddc and $Loadr)
$Loadm	Dollar command that loads data from a GDX file and merges it.
$Loadr	Dollar command that loads data from a GDX file and replaces sets or parameters with the data from that file
$Unload	Dollar command that unloads data to a GDX file.

13.2.2.8 Alter compiler procedures

The dollar commands in this group alter GAMS compilation procedures applied to entries in the input file allowing or disallowing particular syntax choices. The dollar commands in this group are:

$Clearerror	Dollar command that clears compiler errors.
$Dollar	Dollar command that resets character that starts dollar option commands.
$Offdelim	Dollar command that deactivates CSV separation of table data.
$Offembedded	Dollar command that disables the use of embedded values in parameter and set data statements.

© 2022 Prof. Bruce McCarl

$Offempty	Dollar command that prohibits empty data statements.
$Offend	Dollar command that deactivates alternative syntax for flow control statements.
$Offmulti	Dollar command that prohibits multiple data item definitions.
$Offundf	Dollar command that prohibits undf from being assigned.
$Offwarning	Dollar command that activates relaxed domain checking.
$Ondelim	Dollar command that activates CSV separation of table data.
$Onembedded	Dollar command that enables the use of embedded values in parameter and set data statements.
$Onempty	Dollar command that allows empty data statements.
$Onend	Dollar command that deactivates alternative syntax for flow control statements.
$Onmulti	Dollar command that allows multiple data item definitions.
$Onundf	Dollar command that allows undf to be assigned.
$Onwarning	Dollar command that deactivates relaxed domain checking.
$Phantom	Dollar command that designates a phantom set element.
$Use205	Dollar command that tells GAMS to use version 2.05 syntax.
$Use225	Dollar command that tells GAMS to use version 2.25 syntax.
$Use999	Dollar command that tells GAMS to use latest version syntax.
$Version	Dollar command to test GAMS compiler version number.

13.2.2.9 Cause execution of an external program

This dollar commands in this group allow one to execute an external program.

$Call	Dollar command that executes a program during compilation.
$Call.Async	Dollar command that allows asynchronous job handling.
$Call.AsyncNC	Dollar command that allows asynchronous job handling allowing use of multiple processors

13.2.2.10 Restrict access to data

The dollar commands in this group cause GAMS to limit access to data. The dollar commands in this group are:

$Expose	Dollar command that removes all privacy restrictions.
$Hide	Dollar command that hides the objects in a privacy setting but allows them to be used in model calculations.
$Protect	Dollar command that does not allow the objects to be modified in a privacy setting but allows use in model calculations.
$Purge	Dollar command that removes the objects and all data associated in a privacy setting.

13.2.2.11 Tear apart strings

The dollar commands in this group cause GAMS to disassemble strings into multiple environment variables. The dollar commands in this group are:

$Setcomps	Dollar command that dissembles period delimited item into individual components.
$Setnames	Dollar command that tears apart a file name into components.

© 2022 Prof. Bruce McCarl

13.2.2.12 Compress and encrypt files

Input file compression and decompression are available to all users. Encryption and secure work files require special licensing. Three Dollar Control Options control this:

$Compress	<source> <target>	Causes compression of an input file
$Decompress	<source> <target>	Causes decompression of an input file
$Encrypt	<source> <target>	Causes encryption of an input file

13.2.2.13 Handling and definition of macros

The dollar commands in this group are about definition and handling of macros.

$Macro	Dollar command that defines a new macro
$Offdotl	Dollar command that deactivates the automatic addition of .L to variables on the right hand side of calculations
$Offexpand	Dollar command that deactivates the expansion of macros.
$Offlocal	Dollar command that limits use of .local on the same symbol to one in one control stack.
$Offmacro	Dollar command that disables the expansion off macros.
$Ondotl	Dollar command that deactivates the automatic addition of .L to variables on the right hand side of calculations.
$Onexpand	Dollar command that activates the expansion of macros.
$Onlocal	Dollar command that allows unlimited use of .local on the same symbol in one control stack.
$Onmacro	Dollar command that enables the expansion off macros.

13.2.2.14 Include user defined function

GAMS users can define their own functions. Namely functions can be programmed and imported from an external library into GAMS for use in a model.

Function libraries are made available to a model using the $ command `FuncLibIn as follows`:

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

where the

InternalLibName is the name of the library inside your GAMS code.

ExternalLibName is the name that the library has on the disk drive.

In specifying the external file name the default directory is the GAMS systems directory. If the file is to be located some where else then the full path name must be included in the external library name.

When a library of functions is specified then GAMS will validate the library, make the included functions available for use, and add a table of the included functions to the listing file.

An example is presented here.

© 2022 Prof. Bruce McCarl

13.2.3 Detailed description of dollar commands

This section describes each of the dollar commands in detail. The options are listed in alphabetical order.

$$
Abort
Batinclude
Call
Call.Async
Call.AsyncNC
Clear
Clearerror
Comment
Dollar
Double
Drop
Dropenv
Dropglobal
Droplocal
Echo, Echon
Eject
Else, Elsifi, Elseife
Endif
Eolcom
Error
Escape
Eval
Evalglobal
Evallocal
Exit
Expose
Gdxin
Gdxout
Goto
Hidden
Hide
If, If not, Ifi, Ifi not, Ife, Ife not
Ifthen, Iftheni, Ifthene
Include
Inlinecom
Kill
Label
Libinclude
Lines
Load
Loaddc
Loaddcm
Loaddcr
Loadm
Loadr
Log
Macro
Maxcol
Maxgoto
Mincol
Ondelim and Offdelim
Ondigit and Offdigit
Ondollar and Offdollar
Onecho, Onechos, Onechov and Offecho
Onembedded and Offembedded
Onempty and Offempty
Onend and Offend
Oneolcom and Offeolcom
Oneps and Offeps
Onglobal and Offglobal
Oninclude and Offinclude
Oninline and Offinline
Onlisting and Offlisting
Onlocal and Offlocal
Onlog and Offlog
Onmargin and Offmargin
Onmulti and Offmulti
Onnestcom and Offnestcom
Onput, Onputs, Onputv, Offput
Onrecurse and Offrecurse
Onsymlist and Offsymlist
Onsymxref and Offsymxref
Ontext and Offtext
Onuellist and Offuellist
Onuelxref and Offuelxref
Onundf and Offundf
Onverbatim and Offverbatim
Onwarning and Offwarning
Phantom
Prefixpath
Protect
Purge
Remark
Set
Setargs
Setcomps
Setddlist
Setglobal
Setenv
Setlocal
Setnames
Shift
Show
Single
SplitOption
Stars
Stop
Stitle
Sysinclude
Terminate
Title
Unload
Use205
Use225
Use999
Version
Warning

© 2022 Prof. Bruce McCarl

13.2.3.1 $$

Historically all compile time $ commands had to start in column 1.

One can now move them to start elsewhere but needs to add an additional $.

So one can have the sequence (where the first command starts in column 1) as in tranint2.gms

```
$include trandata
*syntax for a $ command not starting in column 1
         $$include tranmodl
         $$include tranrept
```

13.2.3.2 Abort

This issues a compilation error and aborts the compilation placing the associated text message in the LST file. It is employed using the syntax

```
$abort 'text'
```

It can be used with the extension .noerror causing the error count to NOT be increased. When a save file is written, all remaining unexecuted code will be flushed. This allows effective reuse of the save file. The syntax is

```
$abort.noerror 'text'
```

13.2.3.3 Batinclude

The $Batinclude includes an external file with arguments as discussed in the file inclusion chapter. This command is invoked using the syntax

```
$Batinclude filename arg1, arg2,..
```

This by default includes the file from the current working directory but the Idir command line parameter can be used to define a complex search path.

13.2.3.4 Call

This command causes GAMS to call an external program or operating system command during compilation interrupting compilation until the command has been completed. This command is invoked using the syntax

```
$call externalcommand arg1, arg2,..
```

© 2022 Prof. Bruce McCarl

or

```
$call.Async externalcommand arg1, arg2,..
```

Quotes may be placed around the command.

Example:

```
$call 'copy myfile newname'
```

Notes:

- Call is discussed extensively in the Links to Other Programs Including Spreadsheets chapter.
- There is a counterpart command called Execute that is discussed in the Links to Other Programs Including Spreadsheets chapter that operates at execution time.
- Compilation errors are issued if the command or the command processor cannot be loaded and executed properly.
- One can also use the command SplitOption here is one wants to provisionally include the calling parameters for the executable.
- The use without the ASync Suffix means GAMS waits for the job to be completed. When .ASync is included then GAMS proceeds without waiting.
- The command string can be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign.

  ```
  $call 'gams trnsport'
  $call '=gams trnsport'
  ```

  In the second call, the return codes from the system are intercepted and made available to the GAMS system through the errorlevel DOS batch function but they are not in the first.

13.2.3.5 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.

In this cast the operating system starts asynchronous execution sharing the console of the parent process

© 2022 Prof. Bruce McCarl

$CALL.ASYNC "command"

The calls behave identically on Windows and non-Windows platforms.

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. With JobTerminate(pid) or JobKill(pid) an interrupt signal respectively a kill signal can be sent to a running job.

13.2.3.6 Call.AsyncNC

Works like $call amd $call.async 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.

In this cast the NC at the end starts asynchronous calls with a new console rather than sharing the console of the parent process allowing use of multiple processors.

$CALL.ASYNCNC "command"

The calls behave identically on Windows and non-Windows platforms.

13.2.3.7 Clear

This resets GAMS items to their default values. This command is invoked using the syntax

```
$clear item1 item2
```

Notes:

- A list of items follows $clear and results in multiple items being cleared. While the example above lists two items one, two or many more can be listed.
- This is carried out during compile time, and not when the GAMS program executes and it is carried out before any calculations.
- There is an associated option command called clear which operates at execution time. **It is usually the better choice.**
- Not all items can be cleared - only set, parameter, equation and variable types can be reset.
- The result is that sets are emptied and data are zeroed.

13.2.3.8 Clearerror

This clears GAMS awareness of compiler errors. This command is invoked using the syntax

```
$clearerror
```

© 2022 Prof. Bruce McCarl

13.2.3.9 Comment

This changes the character normally used in column 1 to start a comment from * to the single character specified as discussed in the Comment chapter. This command is invoked using the syntax

```
$comment character
```

where character is a one character specification of the new item to use. Note this item should be carefully chosen to not conflict with other usages of the newly chosen character. An example follows (**commentdol.gms**):

```
*normal comment
$comment !
!comment with new character
```

The case of character does not matter when being used.

13.2.3.10 Compress

This causes a file to be compressed into a packed file as discussed in the file compression and encryption chapter. This command is invoked using the syntax

$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

An example is :

```
$compress file.gms compressfile1.gms
```

$Decompress reverse this process.

13.2.3.11 Decompress

This causes a file to be decompressed into an unpacked file as discussed in the file compression and encryption chapter. This command is invoked using the syntax

$Decompress source target

where

source is the name of the compressed source file to be decompressed and

© 2022 Prof. Bruce McCarl

target is the name for the resultant decompressed file

An example is :

```
$decompress compressfile.gms decompressfile.gms
```

$Compress creates the compressed files..

13.2.3.12 Dollar

This dollar command changes the current $ symbol used in dollar sign commands not those in conditionals to the single character specified as c. This command is invoked using the syntax

```
$dollar character
```

where character is a one character specification of the new item to use. The character used should be carefully chosen to not conflict with other usages of the newly chosen character. An example follows (**commentdol.gms**):

```
$Onlisting
$dollar #
#offlisting
```

13.2.3.13 Double

The lines following the $double statement in the echo print of the source file will be echoed in a double spaced fashion in the Echo print within the LST file. This command is invoked using the syntax

```
$double
```

13.2.3.14 Echo, Echon

These options send a text message to an external file. This command is invoked using the syntax

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'other text to be sent' >> externalfile
```

or

```
$echon 'this text' > externalfile
```

or

```
$echon 'a text message' >> externalfile
```

where externalfile is the name including if needed the path of the external file. When the $echon version is used, then no end of line marker is written so the line is repeatedly appended to by subsequent commands. Also note as discussed below $Onecho and $Offecho allows one to copy contiguous lines of text.

© 2022 Prof. Bruce McCarl

Notes:

- Both the text and the file name can be quoted or unquoted.
- The file name by default will go in the working directory.
- The file is not closed until the end of the compilation or when a $call or any kind of $include statement is encountered.
- The symbols > causes any files with the same name to be overwritten.
- The symbols >> causes any files with the same name to be appended to.

13.2.3.15 Eject

This will force a new page to be begun in the LST file. This command is invoked using the syntax

```
$eject
```

13.2.3.16 Encrypt

This causes a file to be converted into an encrypted file as discussed in the file compression and encryption chapter. This command is invoked using the syntax

$Encrypt source target

where

source is the name of the source file to be compressed and

target is the name for the resultant compressed file

An example is :

```
$encrypt file.gms encryptfile.gms
```

A special license is required to use this option as explained in **User's Guide**.

13.2.3.17 Eolcom

This changes the up to 2 character string normally used to delimit an end of line comment from !! to the characters specified as discussed in the Including Comments chapter. This command is invoked using the syntax

```
$eolcom characters
```

where characters are a two character specification of the new delimiter. An example follows (**commentdol.gms**):

© 2022 Prof. Bruce McCarl

```
$Oneolcom
x=x+1;   !! eol comment
$eolcom &&
x=x+1;   && eol comment with new character
```

Notes:

- By default the delimiter is initialized to '!!' but is not active.
- The $eolcom or $Oneolcom dollar command must be used before end of line comments can be employed.
- The $eolcom dollar command sets $Oneolcom to the default setting automatically.

13.2.3.18 Error

Issues a compilation error and continues compilation placing the associated text message in the LST file. It is employed using the syntax

```
$error 'text to include in the LST file'
```

13.2.3.19 Escape

Allows one to print out or display the text sequence for the % syntax used in setting off control variables, system attributes, GAMS command line parameters and arguments in include files. 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 %.

For example (**escape.gms**)

```
$escape &
```

will make %&controlvariable%& print out in a display or put as %controlvariable%. while %&1 will print out as %1. 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.

One may reverse this action with

```
$escape %
```

13.2.3.20 Eval

This dollar command evaluates a numerical expression at compile time and places it into a scoped control variable. In
turn one can use $ife to do numeric testing on the value of this variable. The format is

```
$eval varname expression
```

© 2022 Prof. Bruce McCarl

where the expression must consist of constants, functions or other control variables with numerical values.

For more detailed information see $Eval in the control variables chapter.

13.2.3.21 Evalglobal

This dollar command evaluates a numerical expression at compile time and places it into a global control variable. In
turn one can use $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.

For more detailed information see $EvalGlobal in the control variables chapter.

13.2.3.22 Evallocal

This dollar command evaluates a numerical expression at compile time and places it into a local control variable. In
turn one can use $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.

For more detailed information see $Evallocal in the control variables chapter.

13.2.3.23 Exit

Exits the file currently being utilized. Thus when this is placed in an included file it exits that file but continues in the file where the include appears. In the main program compilation is discontinued at that point. It is employed using the syntax

```
$exit
```

13.2.3.24 Expose

Removes all privacy restrictions from the named item or items.

```
$expose item1 item2
```

or

```
$expose all
```

Notes:

- A list of items follows $expose and results in multiple items being exposed.

© 2022 Prof. Bruce McCarl

- While the statement example above lists two items one, two or many more can be listed.
- The word ALL can also be used to expose all items.
- The Appendix H of the GAMS Users Guide elaborates.
- A special license file is needed for this feature to work.
- The expose only takes effect in subsequent restart files.

13.2.3.25 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

and

```
$Gdxin
```

where the command closes the specified GDX file.

The command must be used in conjunction with the command $Load.

Example:

(gdxintrnsport.gms)

```
$gdxin tran2
$LOAD
  Sets
       uni universal set
       i   canning plants
       j   markets;
$load uni=* i j
  Parameters
       a(i)  capacity of plant i in cases
       b(j)  demand at market j in cases;
$load a=sup
$loaddc b=dem
  Parameter d(i,j)  distance in thousands of miles;
$load d
  Scalar f  freight in dollars per case per thousand miles   ;
$load f
$gdxin
display uni,i,j,a,b,d,f;
```

© 2022 Prof. Bruce McCarl

Notes:

- A Gdxin command followed by the name of the GDX file to use must precede all $Load commands and opens the file.
- A Gdxin command without following arguments must succeed the Load command or commands and closes the file. More than one Load can appear in between.

13.2.3.26 Gdxout

This dollar command is used in a sequence to unload specified items to 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

and

```
$Gdxout
```

where the command closes the specified GDX file.

The command must be used in conjunction with the command $Unload.

Example:

(gdxtrnsport.gms)

```
$gdxout tran
$unload i j
$unload d
$unload f
$unload b=dem a=sup
$gdxout
```

Notes:

- A Gdxout command followed by the name of the GDX file to use must precede all $Unload commands and opens the file.
- A Gdxout command without following arguments must succeed the Unload command or commands and closes the file. More than one Unload can appear in between.

13.2.3.27 Goto

This dollar command will cause transfer of compilation focus to the line starting with an internal label specified through $Label and then continue compilation from there skipping all lines in between. It is employed using the syntax

© 2022 Prof. Bruce McCarl

```
$goto internallabel
```

Notes:

- This dollar command can be used to skip over or repeat sections of the input files.
- In Batinclude files, the target labels or label arguments can be passed as parameters.
- When using a $goto statement GAMS protects against the potential of an infinite loop. The number of times a program jumps back to a label is counted and when a limit is reached, GAMS will issue an error. A maximum of 100 jumps to the same label is the limit.
- Further discussion appears in the Conditional Compilation chapter.

13.2.3.28 Hidden

This dollar command will cause the following text to be treated as a comment that will not be echoed to the listing file as discussed in the Including Comment chapter. It is employed using the syntax

```
$hidden text
```

For example (**commentdol.gms**):

```
$hidden a hidden comment to me
```

13.2.3.29 Hide

This dollar command 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).

```
$hide item1 item2
```

or

```
$hide all
```

Notes:

- A list of items follows $hide and results in multiple items being hidden.
- While the statement example above lists two items one, two or many more can be listed.
- The word ALL can also be used to hide all items.
- Appendix H of the GAMS Users Guide elaborates.
- A special license file is needed for this feature to work.

13.2.3.30 If, If not, Ifi, Ifi not, Ife, Ife not

$If provides control over conditional processing of the input file(s). The syntax is

© 2022 Prof. Bruce McCarl

$If condition statement to execute

or

$If NOT condition statement to execute

or

$Ifi condition statement to execute

or

$Ifi NOT condition statement to execute

or

$Ife condition statement to execute

or

$Ife NOT condition statement to execute

The $Ifi variant is case insensitive while $If is case sensitive and the $Ife variant allows constant expression evaluation.

Notes:

- Numerous conditional expression types can be used as discussed in the Conditional Compilation chapter.
- The result of the conditional test is used to determine whether to include the remainder of the line, which can be any valid GAMS statement including other $ commands like $Goto.
- The first non-blank character on the line following the conditional expression is considered to be the 1st 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 command character, the line is treated as a dollar command line.
- An alternative to placing the statement to execute on the same line as the conditional is to leave the remainder of the line blank and place the statement to execute 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.

Example:

```
scalar a /3/
       b /5/
       c /8/;
```

© 2022 Prof. Bruce McCarl

```
$if a+b==c display a
$ife a+b==c display b
```

a is not displayed because the $if condition is not true while the $ife condition is true and b is displayed.

13.2.3.31 Ifthen, Iftheni, Ifthene, Else, Elseif, Elseifi, Elseife, Endif

These Dollar commands are a form of a $If and control whether a number of statements are active. A $Ifthen/$Iftheni/$Ifthene must be matched with a $Endif. The syntax is generally the same as for the $If statement.

The $Ifthen and $Elseif have variants that are case insensitive ($Ifi and $Elseifi) or evaluate numerical values of the control variables ($Ife and $Elseife).

Notes:

- A 'not' my be used in the commands
- $Else is followed by an instruction which is executed if the matching $ifThen statement is not true.
- $ElseIf / $Elseifi / $Elseife has another comparison behind it.
- One may add a tag to the $Ifthen and $EndIf so it has a definite reference. Then for example $Ifthen.tagone has to match with $Endif.tagone.
- For more information see $Ifthen conditionals

Example:

This example illustrates the use of IFTHEN with and without 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.

© 2022 Prof. Bruce McCarl

13.2.3.32 Include

This includes an external file without arguments as discussed in the Including External Files chapter. This command is invoked using the syntax

```
$include filename
```

This by default includes the file from the current working directory but the Idir command line parameter can be used to define a complex search path.

13.2.3.33 Inlinecom

This dollar command changes the delimiters used to start and end an in line comment from /* and */ to the characters specified. Usage is discussed in the Comments chapter. This command is invoked using the syntax

```
$inlinecom beginningcharacters endingcharacters
```

where beginningcharacters and endingcharacters are each two character specifications of the new beginning and ending delimiters to use. An example follows (**commentdol.gms**):

```
$Oninline
x=x    /* in line comment */ +1;
$inlinecom /& &/
x=x    /& another in line comment &/+1;
```

Notes:

- By default the delimiters are initialized to '/*' and '*/' but are not active.
- The $inlinecom or $Oninline command must be used to activate the end-of-line comment before any in line comments can be used.
- The $inlinecom dollar command activates the ability to use in line comments just as if $Oninline were used.
- Two pairs of character strings must be given.

13.2.3.34 Kill

Removes all data for an identifier with only the type and set definition retained. It is employed using the syntax

```
$kill item1 item2
```

Notes:

- This command should rarely if ever be used. Rather the option command Clear is better.

© 2022 Prof. Bruce McCarl

- A list of items can follow $Kill and if more than one is present results in multiple items being removed. While the statement example above lists two items one, two or many more can be listed.
- 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.
- $Clear has about the same action but the data are treated as if they were zeroed.
- The item in $kill can be redefined.
- More on this appears here.

13.2.3.35 Label

This marks a line to be jumped to by a $Goto statement as discussed in the Conditional Compilation chapter. It is employed using the syntax

```
$label internallabel
```

Notes:

- 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 any $Goto references.

13.2.3.36 Libinclude

The $libinclude includes an external file with arguments as discussed in the Including External Files chapter. This command is invoked using the syntax

```
$libinclude filename arg1, arg2,..
```

This by default includes the file from the inclib subdirectory of the GAMS system directory.

13.2.3.37 Lines

Starts a new page in the echo print part of the LST file if less than n lines are available on the current page. It is employed using the syntax

```
$lines value
```

13.2.3.38 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.

© 2022 Prof. Bruce McCarl

Example:

(gdxintrnsport.gms)

```
$gdxin tran2
$LOAD
  Sets
       uni universal set
       i   canning plants
       j   markets;
$load uni=* i j
  Parameters
       a(i)  capacity of plant i in cases
       b(j)  demand at market j in cases;
$load a=sup
$loaddc b=dem
  Parameter d(i,j)  distance in thousands of miles;
$load d
  Scalar f  freight in dollars per case per thousand miles   ;
$load f
$gdxin
display uni,i,j,a,b,d,f;
```

Notes:

- Load is followed by the names of items to load separated by a space.
- 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 the $Load is not followed by arguments this causes a listing of the GDX file contents to be generated.
- Load brings in the data at compile time and may be used to load sets, parameters, and variable or equation starting values, bounds and scales.
- Execute_load and its variants is the execution time counterpart of this command.
- GAMS does not check to see if the sets referenced match with the elements of the sets in the data. If one wishes that checking then the alternative $Loaddc should be used.
- The universal set can be read for a GDX by using the syntax by `$LOAD id=*.`
- One can use load to define a set based on the elments with non zero entries in the data. thus if one has a set that is to be defined and we know the parameter a is defined over that set containing all the relevant entries then one can use something like the following syntax

```
set i
parameter a(i)
$gdxin trannoset
$load i<a
```

© 2022 Prof. Bruce McCarl

which draws the elements in i from the non zero elements within a (see the example **gdxintrnsportinfersets.gms**)

l

13.2.3.39 Loaddc

This dollar command loads specified items from a GDX file with domain checking. It is employed using the syntax

```
$Loaddc item1 item2 ...
```

and must be used in conjunction with the command $Gdxin.

Example:

(gdxintrnsport.gms)

```
$gdxin tran2
$LOAD
  Sets
       uni universal set
       i   canning plants
       j   markets;
$load uni=* i j
  Parameters
       a(i)  capacity of plant i in cases
       b(j)  demand at market j in cases;
$load a=sup
$loaddc b=dem
  Parameter d(i,j)  distance in thousands of miles;
$load d
  Scalar f  freight in dollars per case per thousand miles   ;
$load f
$gdxin
display uni,i,j,a,b,d,f;
```

Notes:

- 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).
- All other features are the same as discussed under $Load

13.2.3.40 Loaddcm

Dollar command that loads data at compile time plus merges it combining the functions of $Loaddc and $Loadm.

© 2022 Prof. Bruce McCarl

13.2.3.41 Loaddcr

Dollar command that does the same as $Loadr plus domain checking like $Loaddc.

13.2.3.42 Loadm

$Loadm is an alternative form of $load.
It loads specified items from a GDX file. It is employed using the syntax

```
$Loadm item1 item2 ...
```

but must be used in conjunction with the command $Gdxin.
Instead of replacing an item or causing a domain violation error if the item was already initialized it merges the contents.

Example:

In the following example transsol is the GDX file of the model trnsport.gms which can be found in the GAMS Model Lib

```
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
```

13.2.3.43 Loadr

$Loadr is an alternative form of $load.
It loads specified items from a GDX file. It is employed using the syntax

```
$Loadm item1 item2 ...
```

but must be used in conjunction with the command $Gdxin.

It will replace the parameters or sets item1 item2 ... by the data stored in the current GDX file.

Example:

In the following example transsol is the GDX file of the model trnsport.gms which can be found in the GAMS Model Lib

```
sets i / 1*3 /
j / 1*2 /;
$gdxin transsol
$loadr i j
```

© 2022 Prof. Bruce McCarl

```
$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
```

13.2.3.44 Log

This dollar command sends the specified text to the LOG file. It is employed using the syntax

```
$Log text to send
```

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
```

Notes:

- By default, the LOG file is the console.
- The default LOG file can be reset with the Lo and Lf command line parameters.
- Leading blanks are ignored when the text is written out to the LOG file using the $Log command.
- All special % symbols will be substituted out before the text passed through.
- The output goes to the IDE process window but is intermixed with execution reports.

13.2.3.45 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.

For more detailed information see the chapter Macros in GAMS.

© 2022 Prof. Bruce McCarl

13.2.3.46 Maxcol

Sets the right margin for the input file specifying that all valid data is before column n+1 in the input file as discussed in the Including Comments chapter. It is employed using the syntax

```
$maxcol value
```

Notes:

- All text after column value is treated as a comment and is ignored.
- The default value for value is the maximum line length currently 32767.
- Setting maxcol to 0 causes GAMS to set it to the default value.

13.2.3.47 Maxgoto

Dollar command that sets the maximum number of jumps to the same label with $goto. Once the maximum number is reached an error is triggered.

The default setting is 100.

13.2.3.48 Mincol

Sets the left margin for the input file specifying that all valid data is after column n-1 in the input file as discussed in the Including Comments chapter. It is employed using the syntax

```
$mincol value
```

Notes:

- All text before column value is treated as a comment and is ignored.
- The default for value is 1.

13.2.3.49 Ondelim and Offdelim

Controls whether data in table statements are in comma delimited format. It is employed using the syntax

```
$Offdelim
```

or

```
$Ondelim
```

Use of this feature is demonstrated in the example **csv1.gms** and in the file inclusion chapter.

13.2.3.50 Ondigit and Offdigit

Controls the internal precision of numbers. Sometimes a GAMS problem has to be moved from a

© 2022 Prof. Bruce McCarl

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. It is employed using the syntax

```
$Offdigit
```

or

```
$Ondigit
```

Notes:

- If the stated precision of a number exceeds the machine precision an error will be reported. For most machines, the precision is 16 digits.
- The default setting is $Ondigit.
- Ondigit causes GAMS to change numbers to fit machine precision.

13.2.3.51 Ondollar and Offdollar

Controls the echo print of dollar command lines in the LST file. It is employed using the syntax

```
$Offdollar
```

or

```
$Ondollar
```

Notes:

- $Ondollar tells GAMS to include the $commands in the LST file.
- $Offdollar suppresses them.
- The default setting is $Offdollar.

13.2.3.52 Ondotl and Offdotl

Activates or deactivates the automatic addition of .L to variables on the right hand side of calculations as explained below. The syntax is

```
$OndotL
```

or

```
$OffdotL
```

The default is `$OffdotL`.

In report writing one may want to do calculations which reuse terms from the model .. equations in as close to original form as possible. Also one often wished to compute tables of results as a function of the variable levels. Before version 22.9 the only way to do this was with the .l notation as follows (macrotrnsport.gms)

© 2022 Prof. Bruce McCarl

```
zz2=sum((i,j),(x.l(i,j)));
```

where x is a model decision variable.

Today one can use $Ondotl which automatically includes attached the .l to any variables appearing on the right hand side which do not otherwise have a variable attribute extension. In particular the following commands yield the same calculation as that above without the need to the .l's appended to the variable names.

```
$onDotL
zz=sum((i,j),(x(i,j)));
```

This feature was introduced to make macros more useful but is not limited to macros as illustrated above.

This feature once enabled applies to all subsequent instances where variables are on the right hand side of equations and can even be put in the first line of a model.

The command `$offdotl` turns off the implicit .l addition.

13.2.3.53 Onecho and Offecho

Sends multiple subsequent lines to an external file called externalfile. The action is stopped by an $offecho. This command are used employing the syntax

```
$Onecho  > externalfile
line 1 to send
line 2 to send
$Offecho
```

or

```
$Onecho  >> externalfile
line 1 to send
line 2 to send
$Offecho
```

There is also a variant $onechos that permits parameter substitution as $onecho also does and $onechov that forbids parameter substitution.

```
$Onechos  > externalfile
line 1 to send with param sub %it%
line 2 to send
$Offecho
```

Notes

- $Echo allows one to send single lines of text.
- Both the text and the file name can be quoted or unquoted.

© 2022 Prof. Bruce McCarl

- When a path is not included the file by default will be placed in the working directory.
- The file is not closed until the end of the compilation or when a $call or any kind of $include statement is encountered.
- The redirection symbol > causes any files with the same name to be overwritten.
- The redirection symbols >> causes any files with the same name to be appended to.
- Parameter substitution is discussed in the Conditional Compilation chapter.

13.2.3.54 Onembedded and 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. It is employed using the syntax

```
$Offembedded
```

or

```
$Onembedded
```

The default is $Offembedded.

Example:

The element texts for the set i and j will be identical:

```
set k /a,b/
l /a/;
set i(k,l) / a.a 'aaaa cccc dddd', b.a 'bbbb cccc dddd' /
$onEmbedded
set j(k,l) / (a aaaa, b bbbb).(a cccc) dddd /
```

13.2.3.55 Onempty and Offempty

Allows empty data statements for set, parameter or table data. It is employed using the syntax

```
$Offempty
```

or

```
$Onempty
```

Notes:

- $Onempty tells GAMS to allow empty data statements such as (**onempty.gms**)

```
$onempty
set k(*)  an emply set / /;
parameter data(k) / /;
set I /i2/
table aa(I,I)
   i2
i2    ;
```

© 2022 Prof. Bruce McCarl

- $Offempty suppresses them.
- The default setting is $Offempty and data statements cannot be empty.
- One can use the combination of onempty and Onmulti to develop a model without data and add it at a later stage as in **multi.gms**.
- When a set is specified as empty it must be defined with dimensions present ie above the statement **set k(*)** indicates k is one dimensional.
- When a table is specified as empty it must be defined with a place for at least one element as is done in the aa table with the i2 entries above.
- Onempty coupled with Onmulti can be used in conjunction with save and restart to allow a model to be set up and data integrated later possibly to preserve proprietary model structure.

13.2.3.56 Onend and Offend

Activates alternative syntax for flow Control Structures. Namely endloop, endif, endfor, and endwhile are introduced as keywords when the $Onend is active. In turn, these statements end the loop, if, for, and while statements. The dollar command is activated using the syntax

```
$Offend
```

or

```
$Onend
```

Notes

- Both forms of the syntax cannot be valid simultaneously.
- Setting the $Onend dollar command will make the alternate syntax valid, but makes the standard syntax invalid.
- Examples appear in **control.gms**.

13.2.3.57 Oneolcom and Offeolcom

Activates or deactivates use of end-of-line comments as discussed in the Including Comments chapter. By default, the end-of-line comments are set to '!!' but the processing is disabled. The dollar command is employed using the syntax

```
$Offeolcom
```

or

```
$Oneolcom
```

13.2.3.58 Oneps and Offeps

Causes GAMS to treat a zero as an EPS in a parameter or table data statement. The syntax is

```
$Offeps
```

or

© 2022 Prof. Bruce McCarl

```
$Oneps
```

The default setting is $Offeps

13.2.3.59 Onexpand and Offexpand

Activates or deactivates the expansion of macros appearing in the arguments of a macro call. The syntax is

```
$Onexpand
```

or

```
$Offexpand
```

The default is `$Offexpand`.

13.2.3.60 Onglobal and Offglobal

Causes dollar command settings to also be active in included files. The default is that command settings are not inherited ($Offglobal). The syntax is

```
$Offglobal
```

or

```
$Onglobal
```

Dollar command settings specified in the include file will not affect the higher level file.

13.2.3.61 Oninclude and Offinclude

Controls the echo print of included files as discussed in the Including External Files chapter. The default is that statements are included in the echo print ($Oninclude). The syntax is

```
$Offinclude
```

or

```
$Oninclude
```

13.2.3.62 Oninline and Offinline

Activates or deactivates use of in line comments as discussed in the Including Comments chapter. By default, the delimiters are set to '/*' and '*/' but they are not allowed ($Offinline). The syntax is

```
$Offinline
```

or

```
$Oninline
```

© 2022 Prof. Bruce McCarl

13.2.3.63 Onlisting and Offlisting

Activates or deactivates the echo print of input lines to the LST for lines appearing after the $Offlisting. The syntax is

```
$Offlisting
```

or

```
$Onlisting
```

The default setting is $Onlisting.

Notes:

- Suppressed input lines do not generate entries in the symbol and cross-reference sections appearing at the end of the compilation listing.
- Lines with errors will always be listed.

13.2.3.64 Onlocal and Offlocal

$Onlocal allows unlimited use of .local on the same symbol in one control stack while $Offlocal limits the use to one.
The default setting is $Onlocal.

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.

13.2.3.65 Onlog and Offlog

Turns on/off line logging for information about the line number and memory consumption during compilation. This is scoped like the $On/Offisting applying only to included files and any subsequent included files but reverting to `$Onlog` in the parent files.
That means 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.

© 2022 Prof. Bruce McCarl

The syntax is

```
$Offlog
```

or

```
$Onlog
```

The default setting is `$Onlog`.

13.2.3.66 Onmacro and Offmacro

Enables or disables the expansion of macros. The syntax is

```
$Onmacro
```

or

```
$Offmacro
```

The default is `$Onmacro`.

13.2.3.67 Onmargin and Offmargin

Activates or deactivates margin marking as discussed in the Including Comments chapter. The margins are set with $mincol and $maxcol. The syntax is

```
$Offmargin
```

or

```
$Onmargin
```

The default is $Offmargin.

Example:

(margin.gms)

```
$Ontext
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
$Offtext
$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
```

Only the black section of the statements are active since they appear between columns 19 and 45, and anything before 19 or after column 45 is treated as a comment.

© 2022 Prof. Bruce McCarl

This results in the LST file segment

```
7 Now we have          . set i plant /US, UK/      .This defines I
8 turned on the        . scalar x / 3.145 /        .A scalar example.
9 margin marking.      . parameter a, b;           .Define some parameters.
```

where the red dots are the delimiters.

13.2.3.68 Onmulti and Offmulti

Allows or disallows multiple definition and data statements for a set or parameter. By default ($Offmulti), GAMS does not allow data statements to be redefined. If this dollar command is enabled, the second or subsequent data statements are merged with entries of the previous ones. It is employed using the syntax

```
$Offmulti
```

or

```
$Onmulti
```

Ordinarily $Onmulti should not be used as it can have perverse effects.

Example:

```
$Onmulti
Scalar x /3/
Scalar x /4/;
Set I /i1/;
Set I /i2/;
Set j / /;
Set j /a,b,c/;
```

Notes:

- Note that all multiple data statements are executed before any other statement is executed.
- The last value takes precedence.
- Default is $Offmulti.
- One can use the combination of Onempty and onmulti to develop a model without data and add it at a later stage as in **multi.gms**.
- Onmulti coupled with Onempty can be used in conjunction with save and restart to allow a model to be set up and data integrated later possibly to preserve proprietary model structure.

13.2.3.69 Onnestcom and Offnestcom

Allows nesting of in line comments as discussed in the Including Comments chapter. The dollar command is employed using the syntax

© 2022 Prof. Bruce McCarl

```
$Offnestcom
```

or

```
$Onnestcom
```

The default is $Offnestcom

Example:

(Commentdol.gms)

```
$inlinecom { } onnestcom
{ nesting is now possible in comments { braces have to match } }
```

13.2.3.70 OnOrder and OffOrder

Set operations over leads and lags require the subject set to be ordered and predefined. If one wants to use lags and leads on sets that are dynamic and/or are unordered then this $ condition permits that to happen. Namely including the command $offorder removes the requirement.

The dollar command is employed using the syntax

```
$Offorder
```

or

```
$Onorder
```

The default is $Onorder

Example:

(**lagd1.gms**)

An expanded version of the Test Library file lagd1.gms illustrates the use of on/offOrder as follows.

```
set i(*) ;
i("1970")=yes;
i("1975")=yes;
i("1979")=yes;
i("1980")=yes;
alias(i,j);
set linkthem(*,*);
$offorder
linkthem(i,i+1)=yes;
$onorder
display linkthem;
set linkthem2(*,*);
$offorder
linkthem2(i,i++1)=yes;
$onorder
```

© 2022 Prof. Bruce McCarl

```
display linkthem2;
```

- Since the set i is not ordered rather being dynamic with computed values the + and ++ commands would not have worked giving compilation errors. Setting $offOrder allows their use.
- It is probably best to restore the system to normal using $onorder as soon as possible.
- The use of this option comes at a price: the compiler will not be able to diagnose odd and incorrect formulations and data sets.
- This does not allow one to used the ord command with a set with calculated elements.

13.2.3.71 Onput, Onputs, Onputv, Offput

Causes a block of text to be placed in a put file. The offput stops the putting of the text block. The variant Onputs causes parameters in the text block to be substituted while the onputv suppresses substitution. This is illustrated the Output via Put Commands chapter. It is employed using the syntax

```
$Onput
```

or

```
$Onputs
```

or

```
$Onputv
```

Eventually followed by

```
$Offput
```

Notes

- Text from a file can be included in a put file with the Put_utility 'inc' syntax

13.2.3.72 Onrecurse and Offrecurse

Dollar command that allows a file to include itself. The default setting is $Offrecurse.
Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

Example:

a file called file1.gms can contain

```
$onrecurse
$include file1
```

13.2.3.73 OnStrictSingleton and OffStrictSingleton

Controls whether compile time errors are generated when the singleton set specification for a singleton

© 2022 Prof. Bruce McCarl

set has multiple elements. It is employed using the syntax

```
$Offstrictsingleton
```

or

```
$Onstrictsingleton
```

The default is `$Offstrictsingleton` with error messages being generated.

13.2.3.74 Onsymlist and Offsymlist

Controls the incidence in the LST file of the symbol listing. This listing contains the names of all symbols that have been defined and their explanatory text in alphabetical order grouped by symbol type. This is illustrated the Standard Output chapter. It is employed using the syntax

```
$Offsymlist
```

or

```
$Onsymlist
```

The IDE default is $Offsymlist and the command line GAMS one is $Onsymlist.

13.2.3.75 Onsymxref and Offsymxref

Controls the incidence in the LST file of the cross-reference report of all collected symbols in listing file as discussed in the Standard Output chapter. It is employed using the syntax

```
$Offsymxref
```

or

```
$Onsymxref
```

The IDE default is $Offsymxref and the command line GAMS one is $Onsymxref.

13.2.3.76 Ontext and Offtext

The dollar command $Ontext - $Offtext pair encloses comment lines as discussed in the Including Comments chapter. It is employed using the syntax

```
$Ontext
 comment statement 1
 comment statement 2
 …
$Offtext
```

Line numbers in the compiler listing are suppressed for lines enclosed in the $Ontext - $Offtext sequence.

© 2022 Prof. Bruce McCarl

13.2.3.77 Onuellist and Offuellist

This dollar command controls the LST file incidence of a complete listing of all set elements that have been entered. This is illustrated in the Standard Output chapter. It is employed using the syntax

```
$Onuellist
```

or

```
$Offuellist
```

The default is $Offuellist.

13.2.3.78 Onuelxref and Offuelxref

This dollar command controls the incidence of a cross references of set elements in the LST file as discussed in the Standard Output chapter. It is employed using the syntax

```
$Onuelxref
```

or

```
$Offuelxref
```

13.2.3.79 Onundf and Offundf

This dollar command controls the incidence the use of the special value UNDF in data statements and expression as discussed in the Calculating Items chapter. It is employed using the syntax

```
$Onundf
```

or

```
$Offundf
```

The default is Offundf.

13.2.3.80 Onverbatim and Offverbatim

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 >= 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.

Example:

```
$set f 123
$log %f%
$onVerbatim
$log %f%
$offverbatim
```

© 2022 Prof. Bruce McCarl

```
$log %f%
```

The corresponding dmp file contains

```
$log 123
$onVerbatim
$log %f%
$offVerbatim
$log 123
```

13.2.3.81 Onwarning and Offwarning

This option alters the way data domain checking is done. It allows domain errors in data statements that are imported from other systems and reports warnings instead of errors. Internally data are accepted and stored, even though they are outside the domain. The option is employed using the syntax

```
$Onwarning
```

or

```
$Offwarning
```

The default value is $Offwarning.

Notes:

- This switch affects three types of domain error numbers 116, 170 and 171.
- This can have serious side affects and one has to exercise great care when using this feature.

13.2.3.82 Phantom

This dollar command is used to designate a particular element name as a phantom set element. It is employed using the syntax

```
$phantom elementname
```

Example:

```
$phantom null
set i / null/
  j / a,b,null/ ;
display i,j ;
```

The resulting section of the LST file is

```
---- 4 SET I
(EMPTY)
---- 4 SET J
a, b
```

© 2022 Prof. Bruce McCarl

Notes:

- A phantom element is handled like any other set element. 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.
- Note that null does not appear in the listing file.
- Assignment statements on the phantom label are ignored.

13.2.3.83 Prefixpath

A dollar command that augments search path in the Windows path environment variable.

```
$prefixpath value
```

This results in the text in value being appended to the beginning of the search path.

13.2.3.84 Protect

This dollar command 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.

```
$protect item1 item2
```

or

```
$protect all
```

Notes:

- A list of items follows $protect and results in multiple items being protected.
- While the statement example above lists two items one, two or many more can be listed.
- The word ALL can also be used to protect all items.
- Appendix H of the GAMS Users Guide elaborates.
- The protection only takes effect in the restart files.

13.2.3.85 Purge

This dollar command removes the objects and all data associated in a privacy setting.

```
$purge item1 item2
```

or

```
$purge all
```

© 2022 Prof. Bruce McCarl

Notes:

- A list of items follows $purge and results in multiple items being removed.
- While the statement example above lists two items one, two or many more can be listed.
- The word ALL can also be used to remove all items.
- Appendix H of the GAMS Users Guide elaborates.
- A special license file is needed for this feature to work.
- The removal only takes effect in the restart files.

13.2.3.86 Remark

This dollar command adds a comment to the list file with parameter substitution

```
$remark starttext %item% moretext
```

Which if item was a global variable would create a line in the LST file as follows

```
starttext textinitem moretext
```

13.2.3.87 Set and Drop

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein.

```
$set varname value
```

where

varname is any user chosen variable name

value is optional and can contain text or a number

Use of this command is discussed in the Conditional Compilation chapter.

These variables are destroyed using

```
$drop varname
```

13.2.3.88 Setargs

Sets up substitutable parameters as GAMS control variable names.

© 2022 Prof. Bruce McCarl

```
$setargs args
```

For example using

```
$setargs one two thisthree allremain
```

causes a Batinclude file to use one in place of %1, two in place of %2 and thisthree in place of %3 and all remaining arguments are associated with allremain. Thus one could use code like

```
$setargs one two thisthree allremain
%one% = %two% * %thisthree%;
```

in the BATINCLUDE instead of

```
%1=%2*%3;
```

setargs.gms provides an example. Note setargs must appear in the Batincluded file. Also note one can also use a * or a . or a / to cause a numbered item to be skipped over.

For example

```
$setargs * * thisthree *
```

will only put a new name in for %3.

Use of this command is further discussed in the conditional compilation chapter.

13.2.3.89 Setcomps

Establishes or redefines control variables so they contain the components of a period delimited string.

```
$setcomps perioddelimstring v1 v2 v3 ….
```

where perioddelimstring is any period delimited string like the set specification of a multidimensional parameter

- 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

Thus (**condcomp.gms**)

```
$setcomps s1.s2.s3 sel1 sel2 sel3
```

© 2022 Prof. Bruce McCarl

separates the string s1.s2.s3 into its three components placing s1 into sel1, s2 into sel2 and s3 into sel3.

The three items may be recombined back into the original filename string by using %v1%.%v2%.%v3% in a command like (**condcomp.gms**)

```
$setglobal nam1 %sel1%.%sel2%.%sel3%
```

In turn one can do conditional processing as illustrated below

```
scalar count /0/;
set sels /s1*s3/;
loop(sels,count=count+1;
   if(sameas(sels,"%sel2%"),display "found element %sel2% in position",count;);
     );
```

13.2.3.90 Setddlist

Causes GAMS to look for misspelled or undefined 'double dash' GAMS parameters. For example, in the program below the double dash' options in use are 'one', 'two', 'three' and 'four' (note the use of the %two % in quotes automatically makes it part of the allowed list of double dash parameters):

```
$if NOT set one $set one default value
display '%two%';
$setddlist three four
```

The following GAMS invocation will cause an error since --five is not a valid 'double dash' option.

```
gams ein.gms --two=twovalue --five=20
```

13.2.3.91 Setglobal and Dropglobal

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein.

```
$setglobal varname value
```

where

varname is any user chosen variable name

value is optional and can contain text or a number

Use of this command is discussed in the conditional compilation chapter.

© 2022 Prof. Bruce McCarl

These variables are destroyed using

```
$dropglobal varname
```

13.2.3.92 Setenv and Dropenv

Establishes or redefines contents of an environment variable.

```
$setenv varname value
```

where

varname is a user or system environment variable name

value is optional and can contain text or a number

Use of this command is discussed in the conditional compilation chapter.

These variables are destroyed using

```
$dropenv varname
```

13.2.3.93 Setlocal and Droplocal

Establishes or redefines contents of a control variable that is accessible only in the code module where defined.

```
$setlocal varname value
```

where

varname is any user chosen variable name

value is optional and can contain text or a number

Use of this command is discussed in the conditional compilation chapter.

These variables are destroyed using

```
$droplocal varname
```

© 2022 Prof. Bruce McCarl

13.2.3.94 Setnames

Establishes or redefines three control variables so they contain the drive subdirectory, filename and extension of a file named with full path.

```
$setnames filename v1 v2 v3
```

where filename is any file name

- v1 is the name of a control variable that will contain the name of the subdirectory where the file is located
- v2 is the name of a control variable that will contain the root name of the file
- v3 is the name of a control variable that will contain the extension of the file

Thus (**condcomp.gms**)

```
$setnames d:\gams\xxx.txt filepath filename fileextension
```

separates the filename d:\gams\xxx.txt into its three components placing d:\gams\ into filepath, xxx into filename and .txt into fileextension.

The three items may be recombined back into the original filename string by using %v1%%v2%%v3% in a command like (**condcomp.gms**).

```
$setglobal name %filepath%%filename%%fileextension%
```

13.2.3.95 Shift

Shifts the order of all parameters passed once to the 'left'. This effectively drops the lowest numbered parameter in the list. It is employed using the syntax

```
$shift
```

You can use it to process parameters one at a time until you have done them all in a Batinclude context (**shift.gms**, **processshift.gms**).

13.2.3.96 Show

Shows current values of the control variables plus a list of the macros. It is employed using the syntax

```
$show
```

13.2.3.97 Single

Causes all subsequent lines in the echo print source file portion of the LST file being single spaced. It is employed using the syntax

```
$single
```

© 2022 Prof. Bruce McCarl

This dollar command is the default GAMS state, and is only useful as a switch to turn off the $double dollar command.

13.2.3.98 SplitOption

This dollar control splitOption splits a parameter sequence in the form "**-a3=0**" or "**-a2=3.14**" or "**/opt=val**" or "**/opt val**" into two parts placing then into two control variables. This is designed for use in BATINCLUDE statements where one wants to alter parameter when invoking other executables.

The syntax is splitOption PairToSplit ControlName1 ControlName2 where

PairToSplit is a string formatted as -opt=val or -opt val or \opt=val or \opt val

ControlName1 is the name of a compile-time control variable that will contain the name of the parameter to change

ControlName2 is the name of a compile-time control variable that will contain the value of the option.

For example

$splitoptin -pw 200 pgwidthI pgwidth

will return pgwidthI with the text pw and pgwidth with 200

More details on this are given here.

13.2.3.99 Stars

Alters the '****' marker in the GAMS listing file. By default, important lines like those denote errors, and the solver/model status are prefixed with '****'. The syntax to use the command is

```
$stars characters
```

where the characters are the replacement string.

13.2.3.100 Stop

Stops program compilation without creating an error.

```
$stop
```

13.2.3.101 Stitle

Sets a subtitle that is placed in the page header of the LST file as discussed in the Standard Output chapter. The next output line will appear on a new page in the listing file. It is employed using the syntax

```
$stitle 'new title'
```

© 2022 Prof. Bruce McCarl

13.2.3.102 Sysinclude

The $sysinclude includes an external file with arguments as discussed in the Including External Files chapter. This command is invoked using the syntax

```
$sysinclude filename arg1, arg2,..
```

This by default includes the file from the GAMS system directory.

13.2.3.103 Terminate

Dollar command that terminates compilation and execution immediately.

13.2.3.104 Title

This dollar command sets a title that is placed in the page header of the listing file to 'new title' as discussed in the Standard Output chapter. The next output line will appear on a new page in the listing file. It is employed using the syntax

```
$title 'new title'
```

13.2.3.105 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.

Example:

(gdxtrnsport.gms)

```
$gdxout tran
$unload i j
$unload d
$unload f
$unload b=dem a=sup
$gdxout
```

Notes:

- Unload is followed by the names of items to load separated by a space.
- 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.
- Unload outputs the data at compile time and will write the data present at the time that the statement is encountered during the compilation. The results of calculations and solves will not be

© 2022 Prof. Bruce McCarl

reflected.

- Unload should not ordinarily be used, it is safer to use the execution time counterparts Execute_Unload or Execute_Unloaddi as calculations and solves affect the results.
- The only way to guarantee that the data is current is to use the execution time command or to use a save then restart a file with the dump commands within them.

13.2.3.106 Use205

This dollar command sets the GAMS syntax to that of Release 2.05. This is mainly used for backward compatibility and is employed as

```
$use205
```

13.2.3.107 Use225

This dollar command sets the GAMS syntax to that of Release 2.25. This is mainly used for backward compatibility and is employed as

```
$use225
```

13.2.3.108 Use999

This dollar command sets the GAMS syntax to that of the latest version of GAMS. This dollar command is the default. This is mainly used after $use205 or $use225 and is employed as

```
$use999
```

13.2.3.109 Version

Dollar command to test GAMS compiler version number.

```
$version nnn
```

issues a compilation error if nnn is greater than the current GAMS version.

13.2.3.110 Warning

Dollar command that issues a compilation warning but continues compilation and execution.

13.3 The Option Command

GAMS allows users to employ option commands to change solvers, obtain debugging information, alter selected characteristics of the output, alter solution procedures, change GAMS internal settings, and remove items from memory use.

Basics
Options by function

© 2022 Prof. Bruce McCarl

Description of options

13.3.1 Basics

Options allow users to make run time overrides of a number of internal GAMS settings that are adequate for the most purposes but can be manipulated. The general forms an option statement can take on are

```
option namedoption = integer;
option namedoption = real number;
option namedoption = text;
option namedoption;
option itemname:decimals:rowentries:colentries;
```

where

namedoption is one of the option names as discussed below
itemname is the name of an item to be formatted for display statements as discussed below
but more completely in the Report Writing chapter.
the pink item is a setting for the option but in some cases is not required.

Notes:

- Option commands are processed at execution time unlike the Dollar Control commands discussed in the chapter on Dollar Commands.
- More than one option can be included on a line separated by commas or end-of-line characters.
- Option or Options can be used interchangeably.
- Option names are not reserved words and therefore do not conflict with other uses of the same name.
- 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 for all subsequent but no prior instructions.

13.3.2 Options by function

Options can be divided into a number of broad classes. Namely options which

Control Solver Choice
Add debugging output to the LST file
Alter LST file contents
Influence procedures used by solvers
Change other GAMS settings
Eliminate items from memory
Form projections of data items

© 2022 Prof. Bruce McCarl

Below we list the options for each of these categories and later we provide a more complete discussion of each.

13.3.2.1 Options for control of solver choice

One can change the solver applied to a problem type using the option command associated with the problem type.

Option	Basic Description
CNS	Names CNS solver
DNLP	Names DNLP solver
LP	Names LP solver
MCP	Names MCP solver
MINLP	Names MINLP solver
MIP	Names MIP solver
NLP	Names NLP solver
RMINLP	Names RMINLP solver
RMIP	Names RMIP solver
Subsystems	Lists all solvers and current default solvers in LST File.

13.3.2.2 Options including debugging information in LST file

One can add debugging information to the LST file using the option command.

Option	Basic Description
Dmpsym	Gives data on number of cases stored (memory use) for all GAMS items.
Profile	Controls inclusion of statement execution time and memory use information.
Profiletol	Controls minimum execution time for inclusion of a statement in profile output.
Sysout	Adds solver status output file to LST file.

13.3.2.3 Options influencing LST file contents

One can change the contents of the LST file using various option commands.

Option	Basic Description
Option itemname:d	Display item formatting.
Option itemname:d:r:c	Display item formatting.
Decimals	Controls default decimal places in displays.
Dispwidth	Controls width of labels in display statements in the columns. Expands beyond default of 10.
Dmpsym	Gives data on number of cases stored (memory use) for all GAMS items.
Eject	Inserts a page break in the LST file.
Profile	Includes statement execution time and memory use

© 2022 Prof. Bruce McCarl

	information in output file.
Profiletol	Specifies minimum execution time for inclusion of a statement in profile output.
Solprint	Suppresses solution printout in LST file.
Solslack	Includes slacks in solution output.
Sysout	Adds solver status file to LST file.

13.3.2.4 Options influencing solver function

One can change solver limits or the information GAMS passes to the solvers using various option commands.

Option	Basic Description
Bratio	Controls basis formation.
Domlim	Maximum number of domain errors.
Iterlim	Maximum number of solver iterations.
Optca	Absolute optimality tolerance in a MIP.
Optcr	Relative optimality tolerance in a MIP.
Reslim	Maximum seconds job can execute.
Savepoint	Controls construction of saved solution GDX file.
Solprint	Suppresses solution printout in LST file.
Solslack	Includes slacks in solution output.
Solveopt	Controls handling of solution when merged into stored solution.
Sysout	Adds solver status file to LST file.
Threads	Controls number of cores used by solver

One can also use the model attribute `defPoint`.

13.3.2.5 Other options altering GAMS settings

One can change the way GAMS performs certain tasks using various option commands.

Option	Basic Description
Bratio	Controls basis formation.
Decimals	Controls default decimal places in displays.
Forlim	Maximum number of executions of for, repeat or while.
Iterlim	Maximum number of solver iterations.
Oldname	Causes GAMS to only allow 10 character set element names for compatibility with systems that do not accept longer names (notably older versions of MPSGE).
Reslim	Maximum seconds a job can execute.
Savepoint	Controls construction of saved solution GDX file.
Seed	Random number seed.
Solvelink	Controls whether GAMS program stays open during a solve and iln cases the method used when passing information to a solver (in core or out).
Solveopt	Controls handling procedures when solution is merged into stored solution.
Strictsingleton	Controls handling of singletons with multiple elements.

© 2022 Prof. Bruce McCarl

Sys10	Controls handling procedures when exponentiation is done.

13.3.2.6 Options affecting data for items in memory

One can remove the data from an item using two option commands.

Option	Basic Description
Clear	Zeros all data for an item.
Kill	Removes all data for an item.

13.3.2.7 Options that form projections of data items

An option command exists which allows one to rapidly count the number of elements in a particular slice of a parameter. See the description below.

13.3.3 Description of options

Here we discuss each of the options in detail. All excepting the first two will be listed in alphabetical order. The option settings in a model can be listed using DMPOPT.

Option itemname:d and Option itemname:d:r:c
Option itemname < or <= itemname2
Bratio
Clear
CNS
Decimals
DNLP
Domlim
Dmpopt
Dmpsym
Dualcheck
Eject
Forlim
IntVarUp
Iterlim
Kill
Limcol
Limrow
LP
MCP
Measure
MINLP
MIP
NLP
Oldname
Optca
Optcr
Profile
Profiletol
Reslim
RMIP
RMINLP
Savepoint
Seed
Shuffle
Solvelink
Solver
Solprint
Solslack
Solveopt
Subsystems
Sys10
Sysout
Threads

13.3.3.1 Option itemname:d and Option itemname:d:r:c

This option specifies the characteristics of display statement formats as discussed in the Report Writing chapter. When using this option

Itemname	is the name of a GAMS item that can be displayed.
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 statement can be used without the r and c arguments. Note a Display statement is needed to

© 2022 Prof. Bruce McCarl

output the item and that this formatting is used for all subsequent displays of that item.

13.3.3.2 Option itemname < or <= itemname2

An option command exists which allows one to rapidly count the number of elements in a particular slice of a parameter. The general format of this command is

```
Option item1 < item2 ;
```

or

```
Option item1 <= item2 ;
```

Where item1 and item2 are GAMS sets or parameters with conforming domain declarations. The dimensionality of item1 has to be equal or less than the dimensionality of item2. If the item1 dimensionality is less than the item2 dimension, 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. If a symbol has identical domain definitions they are permuted right to left (if < is used) or left to right (if <= is used).

Examples:

Suppose we have Q(I,J,K) and want to know how many elements exist for a set element I across all combinations of the subscripts J and K. This can be done using the option command (**project.gms**)

```
set i /1*3/
    j /1*3/
    k /1*3/;
Parameter Q(I,J,K) / 1.1.1 1, 1.2.3 3, 2.1.1 4/   ;
Parameter Elementcount(I) ;
option elementcount< Q  ;
display elementcount;
```

Whereupon the elementcount parameter would contain a count of the number of nonzero elements in Q associated with each element of the set I. Similarly one could develop a count of the number of nonzero entries within Q for each pairing of the elements J and K across all values of the subscript I. by inserting

```
Parameter Elcount(J,K) ;
option elcount< Q  ;
display elcount
```

This also works for sets (**project.gms**)

```
Set  i, fromto(i,i), tofrom(i,i);
alias(i,ii);
parameter in(i),out(i);
option tofrom < fromto, in < fromto, out <= fromto;
```

which is equivalent to

```
tofrom(i,ii) = fromto(ii,i);
in(i) = sum(fromto(ii,i),1);
out(i) = sum(fromto(i,ii), 1);
```

© 2022 Prof. Bruce McCarl

13.3.3.3 Bratio

This option specifies what GAMS will do in forming an advanced basis as discussed in the Basis chapter. This option is used by setting

```
Option Bratio=realnumber;
```

The value specified for this option causes a basis to be discarded if the number of basic variables is smaller than bratio times the number of equations.

Notes:

- Setting bratio to 1 will always cause the basis to be discarded, which is sometimes needed with nonlinear problems as discussed in the NLP and Basis chapters.
- Setting bratio to 0 forces GAMS to always try to construct a basis.
- 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.
- The allowable values range from 0 to 1 with a default value of 0.25.

13.3.3.4 Clear

This option tells GAMS to resets the named GAMS items to their default values. This command is invoked using the syntax

```
Option Clear=itemname;
```

For an example see **memtest.gms**

Notes:

- This is carried out during execution.
- Not all items can be cleared - only set, parameter, equation and variable types can be reset.
- The result is that the specified set is emptied or all data for the specified parameter are zeroed.
- The Kill option is related but clears out the data and removes all values and should not typically be used.
- Memory space is not recovered unless the job is saved and restarted.

13.3.3.5 CNS

This option specifies what solver GAMS will use when it needs to solve a CNS type of model. This option is used by setting

© 2022 Prof. Bruce McCarl

```
Option CNS=solvername;
```

where the solver must be CNS capable.

13.3.3.6 Decimals

This option specifies the default number of decimal places to be printed by all subsequent display statements. However this specification is not applied to named items having specific display formatting options defined as discussed above or in the report writing chapter. This option is used by setting

```
Option DECIMALS=number;
```

Nonlinear solvers have difficulty recovering after attempting an undefined operation. The default value is 3 and the number can range from 0 to 8.

13.3.3.7 Dispwidth

This option specifies the number of characters to be printed in the column labels of all subsequent display statements when the column labels are of length greater than 10. This option is used by setting

```
Option DISPWIDTH=number;
```

The default value is 10 and the number can range from 10 to 31.

13.3.3.8 DNLP

This option specifies what solver GAMS will use when it needs to solve a DNLP type of model. This option is used by setting

```
Option DNLP=solvername;
```

where the solver must be DNLP capable.

13.3.3.9 Domlim

This option specifies the maximum number of allowable domain errors (undefined operations like division by zero) during a nonlinear solver run before the solver terminates the run. Such errors are encountered while calculating the nonlinear user defined nonlinear terms function and their derivatives as discussed in the Execution Errors chapter. This option is used by setting

```
Option Domlim=number;
```

Nonlinear solvers have difficulty recovering after attempting an undefined operation. The default value is 0.

© 2022 Prof. Bruce McCarl

13.3.3.10 DmpOpt

This option causes GAMS to generate a list of all available options which can be set using the option statement, and their values to the LST file. This option is used by setting

```
Option DmpOpt;
```

13.3.3.11 Dmpsym

This option causes GAMS to generate a dump of the cases stored used by each named item in the GAMS program. It can be used in diagnosing memory problems as discussed in the memory chapter. This option is used by setting

```
Option Dmpsym;
```

13.3.3.12 Dualcheck

This option causes GAMS to evaluate and provide output on the reduced cost condition for each variable in the Limcol output using the row marginals. This option is used by setting

```
Option Dualcheck=1;
```

The default value is no dual check.

13.3.3.13 Eject

This option causes GAMS to inject a page break into the LST file. This option is used by setting

```
Option Eject;
```

13.3.3.14 Forlim

This option 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. This option is used by setting

```
Option Forlim=number;
```

as illustrated in **otheroptions.gms**. The default value is 999999999.

13.3.3.15 IntVarUp

The default upper bound on integer variables has changed as explained here. This can be changed with a command line parameter of with an option statment of the formt:

```
Option IntVarUp=n;
```

using the values discussed under the command line parameter

© 2022 Prof. Bruce McCarl

13.3.3.16 Iterlim

This option specifies the maximum number of allowable solver iterations, before the solver terminates the run. This option is used by setting

```
Option Iterlim=number;
```

As of version 23.1 the default iteration limit has been increased from 10000 to 2e9. Setting `IterLim` to `INF` will not work since it is treated as an integer by GAMS and many solvers. Some solver, e.g. GAMS/Gurobi, recognize 2e9 and set the solver iteration limit to infinity.

13.3.3.17 Kill

This option tells GAMS to remove all data for a named GAMS item. This command is invoked using the syntax

```
Option Kill=itemname;
```

For an example see **memtest.gms**.

Notes:

- Kill should not ordinarily be used. Rather one should use Clear but they operate essentially the same..
- This is carried out during execution.
- Not all items can be killed - only set, parameter, equation and variable types can be reset.
- The result is that only the name and set dependency is retained.
- The Clear option is related but zeros all out the data.
- Memory space is not recovered unless the job is saved and restarted or a solve occurs with solvelink=0.

- More on this appears here.

13.3.3.18 Limcol

This option specifies the number of cases output in the LST file for each variable as discussed in the Standard Output chapter. This option is used by setting

```
Option Limcol=number;
```

The default value is 3.

13.3.3.19 Limrow

This option specifies the number of cases output in the LST file for each named equation as discussed in the Standard Output chapter. This option is used by setting

© 2022 Prof. Bruce McCarl

```
Option Limrow=number;
```

The default value is 3.

13.3.3.20 LP

This option specifies what solver GAMS will use when it needs to solve a LP type of model. This option is used by setting

```
Option LP=solvername;
```

where the solver must be LP capable.

13.3.3.21 MCP

This option specifies what solver GAMS will use when it needs to solve a MCP type of model. This option is used by setting

```
Option MCP=solvername;
```

where the solver must be MCP capable.

13.3.3.22 Measure

This option tells GAMS to output the time and memory use since the last measure statement or the program beginning. This option is used by setting

```
Option Measure;
```

Profile is probably the option to use if one really wants execution timing.

13.3.3.23 MINLP

This option specifies what solver GAMS will use when it needs to solve a MINLP type of model. This option is used by setting

```
Option MINLP=solvername;
```

where the solver must be MINLP capable.

13.3.3.24 MIP

This option specifies what solver GAMS will use when it needs to solve a MIP type of model. This option is used by setting

```
Option MIP=solvername;
```

© 2022 Prof. Bruce McCarl

where the solver must be MIP capable.

13.3.3.25 NLP

This option specifies the solver GAMS will use when it needs to solve a NLP type of model. This option is used by setting

```
Option NLP=solvername;
```

where the solver must be NLP capable.

13.3.3.26 Oldname

This option causes GAMS to only allow 10 character item set element names for compatibility with systems that do not accept longer names (notably older versions of MPSGE). It causes GAMS to check that all set element names are less than 10 characters and uppercases the names.

```
Option Oldname=1;
```

When the set names are too long an execution error arises. There is also an Note it does not require less than 10 character set, parameter, variable etc names and may still not work with older versions.

13.3.3.27 Optca

This option specifies an absolute termination tolerance for use in solving MIP problems. The solver will stop the solution process when a solution is found whose objective value is guaranteed to be within optca of the best possible solution as discussed in the MIP chapter. This option is used by setting

```
Option  Optca=realnumber;
```

The default realnumber is 0.0 but the optcr choice below is used.

13.3.3.28 Optcr

This option specifies a relative termination tolerance for use in solving MIP problems. The solver will stop the solution process when the proportional difference between the solution found and the best theoretical objective function is guaranteed to be smaller than optcr as discussed in the MIP chapter. This option is used by setting

```
Option  Optcr=realnumber;
```

The default realnumber is 0.10.

13.3.3.29 Profile

This option specifies whether to include LST file output on statement level execution timing and memory

© 2022 Prof. Bruce McCarl

use as well as the number of set elements over which statements are executed. Use of this option is discussed in the Speed and Memory chapters. This option is used by setting

```
Option Profile=number;
```

The default value is 0 and a value of

0	Means no execution profile will be generated in the LST file.
1	Means the LST file will contain reports on execution time and memory use for each statement not in a control statement and any first level Control Structure statements like loops, if, for and while.
2	Means that profile information will be reported for any statements that are nested in first level control statements.
3	Profile information is reported for any statements that are nested in second level control statements.

Higher numbers can be used to go further into nested items.

13.3.3.30 Profiletol

This option specifies the minimum amount of time a statement must use to be included in the Profile generated output as discussed in the Speed and Memory chapters. This option is used by setting

```
Option Profiletol=realnumber;
```

The default value is 0.0. Note that profiletol is not applied to model generation statements.

13.3.3.31 Reslim

This option specifies the maximum time in seconds that the computer can run during execution of a solver, before the solver terminates the run. This option is used by setting

```
Option Reslim=realnumber;
```

The default value is 1000.

13.3.3.32 RMIP

This option specifies the solver GAMS will use when it needs to solve a RMIP type of model. This option is used by setting

```
Option RMIP=solvername;
```

where the solver must be RMIP capable.

© 2022 Prof. Bruce McCarl

13.3.3.33 RMINLP

This option specifies the solver GAMS will use when it needs to solve a RMINLP type of model. This option is used by setting

```
Option RMINLP=solvername;
```

where the solver must be RMINLP capable.

13.3.3.34 Savepoint

This option 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. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point Numeric input is expected with the allowable numeric values being

0	no point gdx file is to be saved
1	a point gdx file is to be saved from the last solve in the GAMS model
2	a point gdx file is to be saved from every solve in the GAMS model

The command is implemented with the syntax

```
Option Savepoint=number
```

When Sp=1 the point gdx file saved has the name modelname_p.gdx so for a model identified in the solve statement as transport the file would be transport_p.gdx. On the other hand if Sp=2 then the file name is modelname_pnn.gdx where nn is the solve number as determined internally by GAMS. Thus for a model solved 2 times that is identified with the name firm in the solve statement, then the file names would be firm_p1.gdx and firm_p2.gdx. The file is reloaded with the Execute_loadpoint syntax.

This can also be done through a command line parameter or a model attribute.

13.3.3.35 Seed

This option specifies the seed used for the pseudo random number generator. This option is used by setting

```
Option Seed=number;
```

The default value is 3141. The function Execseed also manipulates and retrieves the random number seed.

13.3.3.36 Shuffle

An option that rearranges the data values within a one dimensional parameter so in a random fashion.

The command is invoked using the syntax

© 2022 Prof. Bruce McCarl

```
Option Shuffle=itemname;
```

Where itemname gives the name of the one dimensional parameter that will be rearranged.

There are four different ways this works depending on the way the parameter is declared. these involve whether it is declared over the universal set or a named set and whether data has been defined for the parameter or not.

In particular if

the parameter is defined over a named set and

- no data are declared then the data in the parameter are filled with randomly assigned numbers that range from one to the number of elements in the set. For example if the parameter is defined over a set with 23 elements then the parameter will be defined with numbers ranging from 1 to 23 with the values randomly arranged
- data are declared then the values are randomly rearranged.

the parameter is defined over the universal set and

- no data are declared then the data in the parameter are filled with randomly assigned numbers that range from one to the number of elements in the universal set. For example if the universal set has 1000 elements then the parameter will be defined for all of the elements in the universal set with numbers ranging from 1 to 1000 with the values randomly arranged.
- data are declared for specified elements then the values associated with those elements are randomly rearranged.

This is implemented in the file shuffleex.gms

Notes:

- Each use of shuffle generates a new random data rearrangement .
- One can use this to generate random rearrangements of a set as illustrated in the shuffleex.gms example for the set j at the bottom. However the set is not rearranged rather a tuple is generated that gives the new order (reorderj in the example)

13.3.3.37 Solvelink

This option controls GAMS function when linking to solvers. The command is implemented with

```
Option Solvelink=number;
```

Where the values for number are

0 in which case GAMS operates as it has for years (default)

© 2022 Prof. Bruce McCarl

1 in which case the solver is called from a shell and GAMS remains open.
2 in which case 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 in which case GAMS starts the solution and continues in a Grid computing environment
4 in which case GAMS starts the solution and wait (same submission process as 3) in a Grid computing environment. This is included mainly for debugging purposes as is solvelink=7.
5 in which case the problem is passed to the solver in core without use of temporary files
6 which when chosen causes GAMS to pass data to the solver incore and not wait for the solver to return the solution. Consequently the solution must be collected later as discussed here. More on this option appears in the command line parameter discussion
7 under which GAMS passes information to the solver in core as it does when when solvelink=6. However in this case GAMS also waits for the solver to return the solution. This is included mainly for debugging purposes as is solvelink=4.

A set of solvelink constants may also be used.

Leaving GAMS open or passing the information in core saves time. On the other hand additional memory is required. This option is best for jobs that have a large data set and solve many small models as in that case one sacrifices memory but avoids the overhead of many GAMS saves and restarts. This is implemented by using the option SOLVELINK that can appear on the command line, as a model attribute or as an internal option statement.

The default setting is zero.

This can also be done through a command line parameter or a model attribute.

13.3.3.38 Solprint

This option controls the printing of the model solution in the LST file as discussed in the Standard Output chapter. This option is used by setting

```
Option Solprint=text;
```

where two text values are allowed

On which includes solution listings following solves.
Off which removes solution listings following solves.
Silent which suppresses all solution information.

The default setting for text is On.

The related model attribute is `<modelname>.solprint=n` and the GAMS parameter is solprint=n .

A related set of solprint constants is also present.

© 2022 Prof. Bruce McCarl

13.3.3.39 Solslack

This option causes the equation output in the listing file to contain slack variable values instead of level values as discussed in the Standard Output chapter. This option is used by setting

```
Option Solslack=value;
```

where two values are allowed

0	which includes equation levels in the solution part of the LST file following solves.
1	which includes equation slacks in the solution part of the LST file following solves.

The default value is 0 so a print out including slacks does not occur.

13.3.3.40 Solveopt

This option controls the way solution values resulting from a solve are stored by GAMS as discussed in the Variables, Equations, Models and Solves chapter. This option is used by setting

```
Option Solveopt=text;
```

where two text values are allowed

Merge	which merges old values with new ones
Replace	which replaces the old values of all equations and most variables.
Clear	which replaces old equation information and all model included variables

The default text is Merge. This is only of concern if a prior solution or starting point is resident in memory and if the sets, variables or equations in the definition of the model vary from the previously solved model or starting point.

Under merge old and new values merged together, and new values overwrite old ones but old nonzero values that do not have new counterparts are left alone. Under replace all old values associated with a variable or equation that at least one instance of appears in a model are reset to default values before new solution values are returned. There are some possible problems with the replace option that users should realize as discussed in **wontgo.pdf** but these are resolved by the clear option.

13.3.3.41 Solver

Option command that makes a solver the default for all model types to which it can be applied.

Set via the command

```
Option solver=abc;
```

For example

© 2022 Prof. Bruce McCarl

```
option solver=conopt;
```

will result into setting the solver for model types LP, RMIP, CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt

```
Additionally

option solver=conopt, solver=cbc;
```

will result into setting the solver for model types LP, RMIP, CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt and then alter the model types that CBC can sollver so CBC is the default like LP, and RMIP, plus add CBC as the default for any model types to which it uniquely applies

13.3.3.42 strictSingleton

This option specifies whether GAMS is to trigger out an error if more than one element is specified in a singleton set. Integer input is expected.

The command is implemented with the syntax

```
Strictsingleton=number
```

Where the values and corresponding actions for the number entry are.

0	Use the first set element ignoring the rest when the singleton set has multiple elements
1	Message out an error if assignment to singleton set causes the set to have multiple elements

The default value is one

13.3.3.43 Subsystems

This option causes GAMS to list all solvers available as well as the current default and active solvers in the LST file. This option is used by setting

```
Option Subsystems;
```

as illustrated in **otheroptions.gms**.

13.3.3.44 Sys10

This option controls whether GAMS converts exponentiation treating a real power as an integer power if the exponent is constant and within 10^{-12} of an integer value. This option is used by setting

```
Sys10=number;
```

© 2022 Prof. Bruce McCarl

where two numerical values are allowed

0	which does not convert exponentiation.
1	which converts exponentiation.

The default value is 0.

there is a corresponding option command implementd by

option sys10=number;

13.3.3.45 Sysout

This option controls the incorporation of additional solver generated output (that in the solver status file) into the LST file. This option is used by setting

```
Option Sysout=text;
```

where two text values are allowed

On	which includes the extra listing following solves.
Off	which excludes the extra listing following solves.

The default text is Off.

The contents of the solver status file can be useful in gaining an understanding of the behavior of the solver. This output is automatically incorporated if the solver crashes or encounters any difficulty.

14 Advanced Language Features

This section covers either very technical or infrequently used features of the GAMS language. The coverage is organized by chapter with the chapters covering:

Output via Put Commands
Acronyms
Conditional Compilation

14.1 Macros in GAMS

GAMS includes the ability to defines macros as of version 22.9. The design of the macro facility was inspired by the GAMS-F preprocessor for function definition developed by Ferris, Rutherford and Starkweather, 1998, 2005.

© 2022 Prof. Bruce McCarl

Macros are widely used in computer science to define and automate structured text replacements. The GAMS macro processors functions similar to the popular C/C++ macro preprocessor. However, it is GAMS syntax driven.

Basic Definition

The definition takes the form

```
$macro name  macro body
$macro name(arg1,arg3,arg2,..) macro body with tokens arg1,..
```

The name of the macro has to be unique, similar to other GAMS data types like sets and parameters. A (following immediately the macro name starts the list of replacement arguments and a) ends it. These will be expanded by the arguments in parentheses in a call to the macro.

The macro body is not further analyzed after removing leading and trailing spaces.

The items to replace in the macro body follow the standard GAMS identifier conventions. For example: let us define a simple macro that forms 1 over an item

```
$macro oneoverit(y) 1/y
```

then let us use it calling the macro twice with two different arguments(macros.gms)

```
z = oneoverit(x1)+oneoverit(x2);
```

will then be expanded using the arguments into:

```
z = 1/x1 +1/x2;
```

as GAMS recognizes **oneoverit(x1)** as a macro and substitutes x1 (the argument in **oneoverit (x1))** for y (in the original definition of the macro **oneoverit(y))**and does the same for x2.

Note the item used in the macro(**y**) is just a symbol and can duplicate the name of other items in the code. For example the macro could have been defined

```
$macro oneoverit(x1) 1/x1
```

even though x1 is a named scalar in the code.

Multiple arguments

The actual calling arguments of macros can contain multiple arguments. In such a case the multiple arguments are separated by commas. (macros.gms)

```
$macro ratio(a,b) a/b
```

when called with

```
z = ratio(x1,x2);
```

will expand into:

```
z = x1/x2;
```

© 2022 Prof. Bruce McCarl

Multi-line Macros

One can extend macros to multiple lines using a \ (macros.gms)

```
$macro equ2(z,d,q)  equation equ2_&z&d; \
      equ2_&z&d.. z*q =e= 0;
```

Spacing within Multi line macros

I recently had problems with a macro and discovered both an issue and a previously hidden GAMS feature that avoided the issue. In particular, suppose I had formed a macro (macrowithampersand.gms) for the right part of the statement below

```
Y= (+12+3*x-0.005*x**2)$(x gt 0 and x <= 200)
```

Where the macro was continued over several lines as follows

```
$macro evalx(x)                        \
            (+12+3*x-0.005*x**2)         \
               $(x gt 0 and          \
                 x <= 200)
```

I then ran this and got an error. Examining the LST file I found the expanded macro was

```
y=(+12+3*x-0.005*x**2)$(x gt 0 andx <= 200);
```

where GAMS had *stripped all beginning and ending spaces* from the lines running the "and" together with the "x" so it was "andx" which the GAMS compiler did not recognize and identified as an error.

I then tried several approaches to fix this and had concluded I could never end a macro line wit and, or, ne, Eq etc. I then asked the GAMS staff if there was a way to avoid the problem and I was informed by Alex that this could be fixed by adding an & at the end of the line after a space as follows

```
$macro evalx(x)                          \
            (+12+3*x-0.005*x**2)       \
               $(x gt 0 and &          \
                 x <= 200)
```

where the & *causes the space to be retained.*

Also in the process I found out that using && caused GAMS to strip quotes from a passed in argument (see the example at bottom of macrowithampersand.gms. Thus if one had the macro above (renamed to evalx1) and called it with

© 2022 Prof. Bruce McCarl

```
Y=evalx1("x");
One would need to modify the macro to

    $macro evalx1(x)                                          \
                    (+12+3* &&x-0.005*&&x**2)
\
                    $(&&x gt 0 and &&x <= 200)
```

I also discovered in forming this example that one could not end a line with an & and then have next line start with && as I would need to do to preserve the spacing after the and command if the macro was typed as above.

Macros within Macros

Macros can be included within macros (macros.gms)

```
$macro product(a,b) a*b
$macro addup(i,x,z) sum(i,product(x(i),z))
```

when called with

```
z = addup(j,a1,x1);
```

will expand into:

```
z = sum(j,a1(j)*x1);
```

Note multiple pairs of parenthesis and quotes can be used freely to protect the separating comma.

More careful expansion

The recognition of macros and expansion of arguments can be more carefully controlled by the use of ampersands (&) in the macro body. Ordinarily the macro will only substitute for full words thus the macro group (macros.gms)

```
$macro  f(i)  sum(j, x(i,j))
$macro equh(q)  equation equ_q(i); equ_q(i).. q =e= 0;
equh(f(i))
```

which would expand to become

```
equation equ_q(i); equ_q(i).. sum(j, x(i,j)) =e= 0;
```

Note this contains q in a number of other places. If one wished to replace some of them as well one could use (macros.gms)

```
$macro  f2(r,i)  sum(j, r(i,j))
$macro equ2(z,d,q)  equation equ2_&z&d; equ2_&z&d.. z*q =e= 0;
equ2(1,(i),f2(x,i))
```

© 2022 Prof. Bruce McCarl

```
equ2(2,(k),f2(r,k))
```

which would expand to become

```
equation equ2_1(i); equ2_1(i).. 1*sum(j, x(i,j)) =e= 0;
equation equ2_2(k); equ2_2(k).. 2*sum(j, r(k,j)) =e= 0;
```

where the &z and $d are replaced.

One can also include expressions with spaces, commas and unbalanced parentheses using && which includes an expression removing the outer set of quotes. (macros.gms)

```
$macro d(q) display &&q;
$macro ss(q) &&q)
d('"hereit is" , i,k')
d('"(zz"')
z=ss('sum(j,a1(j)');
z=ss('prod(j,a1(j)');
```

where note the d expressions contain quotes, spaces and commas and the ss expression has unbalanced parentheses within the quoted parts.

In turn these expand to become

```
display "hereit is" , i,k;
display "(zz";
z=sum(j,a1(j));
z=prod(j,a1(j));
```

Infinite nested macros

Nested macro use can result in an expansion of infinite length. For example:

```
$macro a  b,a
display a;
```

will expand into:

```
display b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,.....
```

GAMS will eventually refuse to do more substitutions and issue a compilation error.

Use in report writing

Another feature of macros is the implicit use of the .L suffix in report writing and other data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code illustrates this feature (macrotrnsport.gms)

```
$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/ ;
```

© 2022 Prof. Bruce McCarl

```
  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 into:

```
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 `$ondotl` enables the implicit `.L` suffix for variables. This feature was introduced to make macros more useful and is not limited to be used in macro bodies. Since this a new feature it has to be enabled. The matching `$offdotl` will disable this feature.

Other notes

Three more commands are relevant to macros.

`$show` will list any GAMS macros defined.

`$onmacro/$offmacro` will enable or disable the expansion of macros; the default is `$onmacro`.

`$on/offexpand` will change the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onexpand` enables the recognition and expansion of macros in the macro argument list. `$offexpand` will restore the default behavior.

Macro definitions are preserved in a save/restart file and are available again when performing a continued compilation.

14.2 Output via Put Commands

Users can find that GAMS displays are inadequate for output presentation. A more customized output can be created using GAMS put commands. However, with this control comes a cost. Put commands involve an increased degree of technical programming.

Basics of put

© 2022 Prof. Bruce McCarl

Details on put related commands
Putting out a block of text: $onput, $offput, $onputs, $onputv
Making puts conditional
Output to other programs
Errors that arise during puts

14.2.1 Basics of put

The basic structure of the put instruction in its simplest form is:

```
put item;
```

where item is any type of output such as explanatory text, labels, parameters, variable attributes, equation attributes or model attributes. However, in order that GAMS direct the output to the appropriate place, the user must first specify the name of output file then issue a command activating that file. Thus a more general put file sequence is:

```
file localfileidentifier /externalfilelocation/;
put localfileidentifier ;
put item1;
put item2;
put item3;
…
```

In this basic structure, the lines

- Defines the file which will receive the output from the put commands giving it a localfileidentifier (an internal item name) and an external file name possibly containing a path location.
- Issues a put statement with the localfileidentifier and nothing else on the line which assigns the **defined files as the current one to which all subsequent puts will be written** until another internal file is referenced.
- Lastly, the subsequent lines containing put commands each write to the current file.
- Text from a file can be included in a put file with the Put utility 'inc' syntax
- Multiple lines of text can be included in a put file with the $onput syntax

Example:

For illustration we specify an example in the context of the transportation model using the file **putex1.gms**. The component of this file involving put commands is as follows:

```
file myputfile;
put myputfile;
put 'Run on ' system.date '  using source file  ' system.ifile ///;
put 'Run over scenario set ' scenarios.ts //;
loop(scenarios,
    Need(Destinaton)=demandscen(destinaton,scenarios);
    Solve tranport using LP minimizing totalcost ;
```

© 2022 Prof. Bruce McCarl

```
    report("total","cost",scenarios)=totalcost.l;
    report("demand shadow price",Destinaton,scenarios)
           = demandbal.m(Destinaton);
    report("supply shadow price",Source,scenarios)
         = Supplybal.m(Source);
                 savtransport(Source,Destinaton,scenarios)
=transport.l(Source,Destinaton);
    put 'Scenario name   ' scenarios.te(scenarios):14
    put ' Optimality status ' tranport.modelstat:2:0 /;
   )     ;
put //;
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
  put @41;
  loop(scenarios,put scenarios.tl:10);
  put /;
  loop(source$sum(scenarios,
        abs(savtransport(Source,Destinaton,scenarios))),
     put 'Incoming From ' source.tl @35;
     loop(scenarios,
       put savtransport(Source,Destinaton,scenarios):10:0);
     put /;
      );
  put 'Quantity demanded ' @35
  loop(scenarios,
      put demandscen(destinaton,scenarios):10:0);
    put /;
    put 'Marginal Cost of meeting demand ' @35
    loop(scenarios,
      put report("demand shadow price",Destinaton, scenarios)
        :10:2);
    put //    );
```

The resultant output is placed on the file myputfile.put and is

```
Run on 01/05/02  using source file  C:\GAMS\GAMSPDF\PutEX1.GMS

Run over scenario set Four alternatives

Scenario name Base Case      optimality status  1
Scenario name No Chicago     optimality status  1
Scenario name No New York    optimality status  1
Scenario name No Topeka      optimality status  1

Report for New York                  ------------  Scenario  ------------
                                       base      scen1     scen2     scen3
Incoming From Seattle                    50       350         0         0
Incoming From San Diego                 275       275         0       525
Quantity demanded                       325       625         0       525
```

© 2022 Prof. Bruce McCarl

```
Marginal Cost of meeting demand      250.00    250.00    250.00    250.00

Report for Chicago                  ----------  Scenario  ----------
                                       base      scen1     scen2     scen3
Incoming From Seattle                   300         0       350       350
Incoming From San Diego                   0         0       275        25
Quantity demanded                       300         0       625       375
Marginal Cost of meeting demand      178.00    178.00    187.00    187.00

Report for Topeka                   ----------  Scenario  ----------
                                       base      scen1     scen2     scen3
Incoming From San Diego                 275       275       275         0
Quantity demanded                       275       275       275         0
Marginal Cost of meeting demand      151.00    151.00    151.00    151.00
```

Notes:

The file specification must always appear before any of the put commands and in general is as follows

```
File localname / externalfilelocation/;
```

More on the file specification appears below.

- The second entry in any put a sequence must always be of the form

```
Put localfileidentifier optional other contents
```

where the localfileidentifier must match that found in an earlier file statement.

- The subsequent put statements can contain
  - Quoted text as discussed below.
  - Set element labels as discussed below.
  - Set element explanatory text as discussed below.
  - GAMS item explanatory text (for sets, parameters, variables, equations, and models) as discussed below.
  - Parameter numeric data as discussed below.
  - Model solution numeric data as discussed below.
  - System information like source file name or date as discussed below.
  - Write position control characters as discussed below.
  - Item width, decimals and justification specifications as discussed below.
- The output is structured assuming a proportional font is being used (like Courier New). Non-proportional font features are not present.

14.2.2 Details on put related commands

The main put related commands and put formatting are discussed by command below.

© 2022 Prof. Bruce McCarl

File
Put

14.2.2.1 File

The file specification must always appear before any put commands and in general is formatted as follows

```
File localfileidentifier optional explanatory text / externalfilelocation/;
```

or

```
Files localfileidentifier optional explanatory text / externalfilelocation/;
```

Notes:

- The externalfilelocation entry can be any valid filename and file location on the computer system including a full path specification. It is limited to 255 characters.
- When the externalfilelocation is left off the external name will be localfileidentifier.put ie the localfileidentifier plus the extension put.
- If the externalfilelocation does not contain a path specification within then by default the file is placed in the directory where one is working.
- The default directory where put files are kept may be altered with the Putdir command line parameter.
- The localfileidentifier and optional explanatory text must obey the GAMS item naming rules as specified in the Name Rules chapter.
- A file statement can contain more than one localfileidentifier and associated location.

```
File name1/"c:\file1.out"/,file2 /"d:\out\myreport.txt"/;
```

- There can be more than one file statement in a program.
- The name of the active put file can be changed using Put_utility

14.2.2.1.1 Putdr: Pdir

When a path name is not given in the file statement the put files are placed in the current working directory. However this can be reset using the command line Putdir or Pdir options that specifies the directory where the put files are generated and saved. This option does not override the paths specified in file statements.

14.2.2.1.2 .Pdir

One can redirect the put file output to the scratch directory by setting the put file attribute

```
filename.pdir=1;
```

© 2022 Prof. Bruce McCarl

14.2.2.1.3 Sending output to the LOG file

One can choose to send the put file output to the LOG file by using a null specification for the file name as follows and in **put11.gms**

```
file name / '' /;
put name ;
put 'instructions that will go to the log file' /;
put 'more instructions that will go to the log file' /
```

14.2.2.1.4 Sending output to the SCREEN

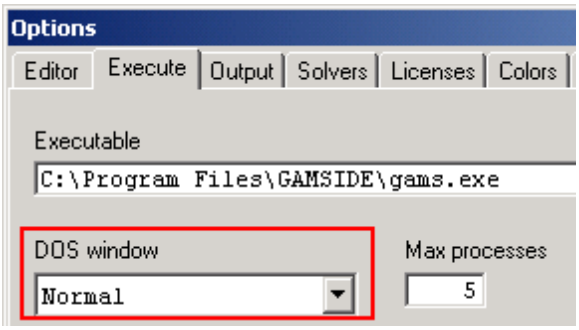
One can choose to send the put file output to the console screen by using the following specification for the file name (**comparewhere.gms**).

```
$set console
$if %system.filesys% == UNIX  $set console /dev/tty
$if %system.filesys% == DOS $set console con
$if %system.filesys% == MS95  $set console con
$if %system.filesys% == MSNT  $set console con
$if "%console%." == "." abort "filesys not recognized";
file screen / '%console%' /;
```

then using put commands like (**comparewhere.gms**).

```
put screen;
put 'I am on scenario ' Scenarios.tl;
putclose;
```

This works fine in DOS or UNIX but not under the IDE. There you need to make the DOS window visible by manipulating the options under the execute tab or just send to the LOG file.



14.2.2.2 Put

The basic syntax for the put command is as follows

```
Put localfileidentifier item1 item2, item3;
```

© 2022 Prof. Bruce McCarl

where localfileidentifier identifies the place to which the put output is to be directed. Item1, item2 and item3 and any other following trailing items are some mixture of data items, labels and formatting commands as discussed immediately below.

- The localfileidentifier entry is not always required as GAMS will direct output to the localfileidentifier found in the last instance of a put localfileidentifier command.
- At least one command of the form put localfileidentifier must appear in the program before any other puts can appear.
- One can have a program which initially puts to one localfileidentifier files and switches to another then switches back as illustrated below (**putex2.gms**)

```
file my1;
file myfilewithalongname;
put my1, 'First line' /;
put myfilewithalongname, 'First line over there' /;
put my1, 'Second line' /;
put myfilewithalongname, 'Second line over there'/;
```

which results in file my1.put with contents

```
First line
Second line
```

and a file named myfilewithalongname.put with contents

```
First line over there
Second line over there
```

- The separators between the items can either be spaces or commas as illustrated above.

14.2.2.2.1 Items within a put

A put may contain items that are quoted text, set element names, set element explanatory text, item names, parameter data, model solution data, GAMS command line parameters, system information, or formatting characters. Each will be reviewed below.

14.2.2.2.1.1 Quoted text

One of the allowable types of items in a put command is quoted text. The blue entries just below all involve instances of quoted text which are used in the example **putex1.gms**

```
put 'Run on ' system.date '  using source file  ' system.ifile ///;
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
  put @41;
  loop(scenarios,put scenarios.tl:10);
```

© 2022 Prof. Bruce McCarl

```
put /;
loop(source$sum(scenarios,
      abs(savtransport(Source,Destinaton,scenarios))),
   put 'Incoming From ' source.tl @35;
   loop(scenarios,
     put savtransport(Source,Destinaton,scenarios):10:0);
   put /;
    );
put 'Quantity demanded ' @35
```

The resultant output file follows with the blue entries corresponding to quoted text generated by the put commands above.

```
Run on 01/05/02  using source file  C:\GAMS\GAMSPDF\PutEX1.GMS

Report for New York                    ------------  Scenario  ------------
                                          base       scen1      scen2      scen3
Incoming From Seattle                       50         350          0          0
Incoming From San Diego                    275         275          0        525
Quantity demanded                          325         625          0        525
```

- Quoted text may be encased in a pair of single (') or double (") quotes with each the items needing to use a matching pair. Thus the following three lines are all exactly equivalent.

```
put 'Run on ' system.date ' using source file ' system.ifile;
put "Run on " system.date " using source file " system.ifile;
put 'Run on ' system.date " using source file " system.ifile;
```

- Quoted text may be specified with a length as discussed below and a justification as discussed below using commands like

```
put 'Marginal Cost of meeting demand ':33 @35);
put 'Marginal Cost of meeting demand ':>33 @35);
```

14.2.2.2.1.2 Set elements

Another of the allowable types of items in a put command is set element related text. These can include the names of the set elements or the explanatory text that is associated with a set element.

Sometimes one wishes output wherein the set element names are used in labeling the printed data. In the put file context this is done by putting an item in a put command which is setname.tl wherein the named set is varied by a loop command as discussed in the Control Structures chapter. The orange entries just below all involve instances where the text for set element names is used in putting outt information within the example **putex1.gms**.

```
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
  put @41;
```

© 2022 Prof. Bruce McCarl

```
loop(scenarios,put scenarios.tl:10);
put /;
loop(source$sum(scenarios,
      abs(savtransport(Source,Destinaton,scenarios))),
   put 'Incoming From ' source.tl @35;
   loop(scenarios,
     put savtransport(Source,Destinaton,scenarios):10:0);
   put /;
    );
```

The resultant output file follows with the orange entries corresponding to set element names generated by the put commands above.

```
Report for New York                  ------------  Scenario  ------------
                                      base       scen1      scen2      scen3
Incoming From Seattle                   50         350          0          0
Incoming From San Diego                275         275          0        525

Report for Chicago                   ------------  Scenario  ------------
                                      base       scen1      scen2      scen3
Incoming From Seattle                  300           0        350        350
Incoming From San Diego                  0           0        275         25

Report for Topeka                    ------------  Scenario  ------------
                                      base       scen1      scen2      scen3
Incoming From San Diego                275         275        275          0
```

Notes:

- The extension .tl is used to cause printing of the text giving the set element name.
- The set must be controlled by a loop statement, GAMS will not automatically cover all cases.
- Default field width is 12 characters but alternative widths may be used as discussed below.
- Default justification is left but alternative formatting may be used as discussed below.
- The set element name capitalization rules follow those discussed in the chapter on Rules for Item Capitalization and Ordering.
- One often finds the names need to be altered or made longer to improve their content. This may be done using the .te syntax discussed just below.

Sometimes one wishes output wherein the explanatory text associated with set element names is used in labeling the printed data. In a put file context this is done by entering an item in a put command which is the setname.te(setname2) wherein the set setname2 is varied by a loop command as discussed in the Control Structures chapter. The blue entries just below all involve instances where the explanatory text associated with set element names are used in putting out information within the example **putex3.gms**. The red entries identify the loop command and the set under control, the orange a case where explanatory set element text is used and the blue a case where the set from which the explanatory set element text is coming differs from the set being varied.

© 2022 Prof. Bruce McCarl

```
set j /a1*a3
        a4 this is element 4
        a5 has a crummy name/;
set i /1,2,3,4 this one is 4/
set newnames(j) /a1 Bolts,a2 Nuts,a3 Cars, a4 Trains, a5 /;
put /
   'Set EL        Explanatory Text        Exp. Text from Subset' //;
loop(j,put j.tl:10 ' !! ' j.te(j):20 ' $$ 'newnames.te(j):20 /);
```

The result with the set element names as output by .tl, the original explanatory set element text and the explanatory set element text coming from a different set which is a subset of the set being varied.

```
Set EL        Explanatory Text      Exp. Text from Subset

a1          !! a1                   $$ Bolts
a2          !! a2                   $$ Nuts
a3          !! a3                   $$ Cars
a4          !! this is element 4    $$ Trains
a5          !! has a crummy name    $$ a5
```

Notes:

- The extension .te followed by the name of the set being varied is used to cause printing of the explanatory text for the set element.
- The set being varied must be controlled by a loop statement, GAMS will not automatically cover all cases.
- Default width is 12 characters but alternative widths may be used as discussed below.
- Default justification is left but alternative formatting may be used as discussed below.
- The set descriptive text capitalization rules follow those discussed in the chapter on Rules for Item Capitalization Ordering.
- It is often useful to modify the set element text to improve their content. This may be done using subsets and the .te syntax as the blue case illustrates.
- Tuples can be used with the .te command not with the .tl. In such cases one gets a printout of either the set elements in the tuple or the text associated with the tuple element depending on the .tf setting as discussed next.
- When .te is used but no explanatory text was entered then by default the set element name is used. This can be changed using the .tf put file attribute where setting the following influences the text used to fill the empty explanatory text field via the rules below as used in **putex5.gms**
  - localfileidentifier.tf to 0 suppresses the fill of missing explanatory text with element names leaving blanks.
  - localfileidentifier.tf to 1 results in blank entries when an element is referenced which does not exist and does the default fill otherwise.
  - localfileidentifier.tf to 2 is the default and always fills empty explanatory text with the element

© 2022 Prof. Bruce McCarl

name.

- localfileidentifier.tf to 3 always fills the .te output with the element names not using the defined explanatory text.
- localfileidentifier.tf to 4 puts out the .te as when .tf=3 in quotes with comma separators. This can be used in generating code for reinclusion in GAMS.
- localfileidentifier.tf to 5 is same as .tf=4 with periods as separators.
- localfileidentifier.tf to 6 is same as .tf=4 with spaces as separators.

- When a tuple is used with .te but no explanatory text is present one gets the names of the set elements defining the tuple separated by a period.

Example:

In the example **putex3.gms** which puts to the localfileidentifier file my1 this is done as follows

```
my1.tf=0;
put / 'Set EL        Explanatory Text        Exp. Text from Subset' //;
loop(j,put j.tl:10 '!!' j.te(j):20 '$$'newnames.te(j):20/);
```

yielding the output

```
Set EL        Explanatory Text        Exp. Text from Subset

a1         !!                      $$ Bolts
a2         !!                      $$ Nuts
a3         !!                      $$ Cars
a4         !! this is element 4    $$ Trains
a5         !! has a crummy name    $$
```

where in comparison with the output just above the entries for a1-a3 in the second column are suppressed as is the one for a5 in the last column due to a lack of explanatory text.

GAMS allows one to put out the fully write out the name and set definition of an indexed item like a parameter, variable, or equation. This is done using the suffix .tn in a put context.

```
put parametername.tn(setdependency)
```

where the set deendency is controlled by loop statements.

Examples (putex1.gms):

Suppose one wants to put out the full identiy of items in an array. One can do this in a transport context as follows

```
put / "example of .tn use" //;
loop((scenarios,Source,Destinaton)$savtransport(Source,Destinaton,scenarios),
     put savtransport.tn(Source,Destinaton,scenarios):0:0 @50
         " = "
         savtransport(Source,Destinaton,scenarios) /;
```

© 2022 Prof. Bruce McCarl

```
);
```

This will produce

```
example of .tn use

savtransport('Seattle',"New York",'base')        =       50.00
savtransport('Seattle','Chicago','base')         =      300.00
savtransport("San Diego","New York",'base')      =      275.00
savtransport("San Diego",'Topeka','base')        =      275.00
savtransport('Seattle',"New York",'scen1')       =      350.00
savtransport("San Diego","New York",'scen1')     =      275.00
```

where note the parameter name and associated set elements are printed out for each nonzero item.

14.2.2.2.1.3 Item explanatory text via .ts

Another of the allowable items in a put command is the explanatory text associated with a named item (a set, file, acronym, parameter, variable, model or equation). These are addressed using the syntax itemname.ts. An example of such addressing occurs in the **putex1.gms** example as follows:

```
put 'Run over scenario set ' scenarios.ts //;
```

where the resultant put file contains

```
Run over scenario set Four alternatives
```

14.2.2.2.1.4 Numeric items

Another of the allowable types of items in a put command is numerical data either for parameters or the attributes of variables, equations or models.

Parameter values
Model solution status attributes: .Modelstat, .Solvestat, .Tmodstat, .Tsolstat
Variable and equation attributes: .L and .M

Numerical parameter data may be included in a put employing the syntax

```
Put parametername(setdependency)
```

where any sets must be controlled in loop statements. The aquamarine entry just below involve instances where the data for a parameter are reported out within the example **putex1.gms**. The red entries identify the loop command and the set under control.

```
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
```

© 2022 Prof. Bruce McCarl

```
put @41;
loop(scenarios,put scenarios.tl:10);
put /;
loop(source$sum(scenarios,
      abs(savtransport(Source,Destinaton,scenarios))),
   put 'Incoming From ' source.tl @35;
   loop(scenarios,
     put savtransport(Source,Destinaton,scenarios):10:0);
   put /;
    );
```

The resultant output file follows with the aquamarine entries corresponding to parameter data generated by the put commands above.

```
Report for New York                ------------  Scenario  ------------
                                     base       scen1      scen2      scen3
Incoming From Seattle                  50         350          0          0
Incoming From San Diego               275         275          0        525

Report for Chicago                 ------------  Scenario  ------------
                                     base       scen1      scen2      scen3
Incoming From Seattle                 300           0        350        350
Incoming From San Diego                 0           0        275         25

Report for Topeka                  ------------  Scenario  ------------
                                     base       scen1      scen2      scen3
Incoming From San Diego               275         275        275          0
```

Notes:

- All sets must be controlled in loop statements, GAMS will not automatically cover all cases.
- Default field width is 12 characters but alternative widths may be used as discussed below.
- Small or large numbers cause exponential format to be used.
- GAMS will print zeros out here exhibiting different behavior than is exhibited in display statements.
- Default number of decimal places is 2 but alternative specifications may be used as discussed below.
- Default justification is right but alternative formatting may be used as discussed below.
- One can suppress small numbers using the .nz specification as the example **putex3.gms** illustrates

```
my1.nz=0.01;
loop(j, put newnames.te(j):12,data(j) /);
```

which would cause any entries with absolute value less than 0,01 to be reported as a zero.

© 2022 Prof. Bruce McCarl

The attributes of models may be included in a put employing the syntax

```
Put modelname.attribute
```

The blue entries below involves the modelstat and solvestat model attributes plus their text counterparts (Tmodstat, Tsolstat) reporting model and solver solution status as used within the example **putex1.gms**.

```
loop(scenarios,
    Need(Destinaton)=demandscen(destinaton,scenarios);
    Solve tranport using LP minimizing totalcost ;
    report("total","cost",scenarios)=totalcost.l;
    report("demand shadow price",Destinaton,scenarios)
            = demandbal.m(Destinaton);
    report("supply shadow price",Source,scenarios)
          = Supplybal.m(Source);
                  savtransport(Source,Destinaton,scenarios)
=transport.l(Source,Destinaton);
    put 'Scenario name   ' scenarios.te(scenarios):14
    put ' Optimality status     ' tranport.modelstat:2:0 /;
    put ' Optimality status text ' tranport.Tmodstat /;
    put ' Solver status         ' tranport.solvestat:2:0 /;
    put ' Solver status text     ' tranport.Tsolstat /;   )      ;
```

The resultant output file follows with the blue entries corresponding to the model solution status attribute data generated by the put commands above.

```
Scenario name   Base Case     Optimality status      1
                              Optimality status text 1 OPTIMAL
                              Solver status          1
                              Solver status text     1 NORMAL COMPLETION
Scenario name   No Chicago    Optimality status      1
                              Optimality status text 1 OPTIMAL
                              Solver status          1
                              Solver status text     1 NORMAL COMPLETION
```

Notes:

- Default field width is 12 characters but alternative widths may be used as discussed below.
- The model solution status attributes that can be used are

Modelstat	which gives problem optimality status as in the modelstat table
Solvestat	which gives solver completion status as in the full list that appears here.
Tmodstat	which gives problem optimality status text as in the modelstat table
Tsolstat	which gives solver completion status text as in the full list that appears here.

- Many more model attributes may be included. A list appears in the Model Attributes chapter.
- Small or large numbers cause exponential format to be used.

© 2022 Prof. Bruce McCarl

- GAMS will print zeros out here but does not in display statements.
- Default number of decimal places is 2 but alternative specifications may be used as discussed below.
- Default justification for text is left but alternative formatting may be used as discussed below.

The value of numerical results in attributes of variables, and equations may be included in a put employing the syntax

```
Put itemname.attribute(setdependency)
```

where any sets must be controlled in loop statements. The colored entries just below involves the .L and .M variable and equation attributes reporting optimal variable levels and equation shadow price marginals as used within the example **putex11.gms**.

```
loop(scenarios,
    Need(Destinaton)=demandscen(destinaton,scenarios);
    Solve tranport using lp minimizing totalcost ;
    report("total","cost",scenarios)=totalcost.l;
    report("demand shadow price",Destinaton,scenarios)
           = demandbal.m(Destinaton);
    report("supply shadow price",Source,scenarios)
         = Supplybal.m(Source);
savtransport(Source,Destinaton,scenarios)=transport.l(Source,Destinaton);
    put 'Scenario name   ' scenarios.te(scenarios):14
    put @30 ' Optimality status     ' tranport.modelstat:2:0 /;
    put @30 ' Optimality status text ' tranport.Tmodstat /;
    put @30 ' Solver status         ' tranport.solvestat:2:0 /;
    put @30 ' Solver status text     ' tranport.Tsolstat /;
put //;
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15 "demand location in "
       scenarios.te(scenarios):0  " scenario" //;
  loop(source$transport.l(Source,Destinaton),
     put 'Incoming From ' source.tl @35;
     put transport.l(Source,Destinaton):10:0;
     put /;
      );
  put 'Quantity demanded ' @35
  put Need(Destinaton):10:0;
  put /;
         put 'Marginal Cost of meeting demand ' @35
         put  demandbal.m(Destinaton):10:2;
         put /
  put /;
    );
put / );
```

© 2022 Prof. Bruce McCarl

Part of the resultant output file follows with the red entries corresponding to the variable level attribute data generated by the put commands above and the blue the equation marginal attribute data.

```
Report for New York       demand location in Base Case scenario

Incoming From Seattle                      50
Incoming From San Diego                   275
Quantity demanded                         325
Marginal Cost of meeting demand        250.00

Report for Chicago        demand location in Base Case scenario

Incoming From Seattle                     300
Quantity demanded                         300
Marginal Cost of meeting demand        178.00
```

Notes:

- Default field width is 12 characters but alternative widths may be used as discussed below.
- Small or large numbers cause exponential format to be used.
- GAMS will print zeros out here but does not in display statements.
- Default number of decimal places is 2 but alternative specifications may be used as discussed below.
- Default justification is right but alternative formatting may be used as discussed below.
- One can suppress small numbers using the .nz specification.
- The variable and equation attributes that can be used are defined in the Variables, Equations, Models and Solves chapter and include

.l	solution level
.m	marginal
.up	upper bound
.lo	lower bound
.scale	scale factor
.prior	variable priority in MIPs

14.2.2.2.1.5 System attributes

Another of the allowable types of items in a put command is the group of **system attributes** such as today's date and time of day, various file names, solver names and model title. The attributes are referenced as follows

```
Put system.attribute;
```

An example using these items appears in the files **putex1.gms** and **putsystem.gms**. The component of this file involving **system attributes** in put commands is as follows:

© 2022 Prof. Bruce McCarl

```
put 'Run on ' system.date '  using source file  ' system.ifile ///;
```

The resultant output is placed on the file myputfile.put and is

```
Run on 01/05/02  using source file  C:\GAMS\GAMSPDF\PutEX1.GMS
```

The complete list of system attributes and their description follows.

.CNS	.MIP	.Rdate
.Date	.MINLP	.Rfile
.Dirsep	.NLP	.RMINLP
.DNLP	.MCP	.RMIP
.Fe	.MPEC	.Rtime
.Fn	.Ofile	.Sfile
.Fp	.Opage	.Sstring
.Gamsrelease	.Page	.TAB
.Gstring	.Pfile	.Time
.Ifile	.Platform	.Title
.Iline	.Prline	.Version
.Lice1 .Lice2		.Prpage
.LP		

Solver that is currently active for CNS problems.

Date on which the program executed.

System attribute that provides access to the file/directory separator character used under this operating system. On Windows platforms this is `\` and on Unix and Mac this is `/`. This helps in writing platform independent GAMS models.

One can use

```
x=%system.dirsep%;
```

or

```
put system.dirsep
```

Solver that is currently active for DNLP problems.

Identifies file extension of input file.

© 2022 Prof. Bruce McCarl

Identifies file name stem of input file.

Dollar command which identifies file path of input file.

A system attribute that gives the version number of the current GAMS release. Gives a value like 22.7

Identifies specific GAMS version being used.

Name of GAMS source input file (GMS file) being executed including storage path.

Number of lines in input file.

GAMS license information.

Solver that is currently active for LP problems.

Solver that is currently active for MIP problems.

Solver that is currently active for MINLP problems.

Solver that is currently active for NLP problems.

Solver that is currently active for MCP problems.

Solver that is currently active for MPEC problems.

Name of GAMS output (Lst) file being used including storage path.

Number of the output page.

Page number of the current page being written.

Put file name for currently active file.

Computer operating system information.

Output line number.

Output page number.

Date information from the GAMS restart file being used.

© 2022 Prof. Bruce McCarl

Name of the GAMS restart file being used including storage path.

Solver that is currently active for RMINLP problems.

Solver that is currently active for RMIP problems.

Time of day information from the GAMS restart file being used.

Name of the GAMS save file being used including storage path.

Identifies name of last solver used.

Time of day when the program was executed.

Title used in the $title command for this file.

GAMS version being run.

14.2.2.2.1.6 GAMS command line parameters

Another of the allowable types of items in a put command is the group of GAMS command line parameters such as input file name and page size. The attributes are referenced as follows

```
Put "%GAMS.commandparameter%";
```

where we are really addressing the text string in the command line parameter as discussed in the Conditional Compilation chapter. An example using these items appears in the files **putex1.gms**. The component of this file involving **command line parameters** in put commands is as follows:

```
put "page size         = " "%gams.ps%" /;
put "gams input file   = " "%gams.input%" /;
put "gams restart file = " "%gams.restart%" /;
```

The resultant output is placed on the file myputfile.put and is

```
page size         = 999
gams input file   = C:\GAMS\GAMSPDF\BIGONE\PutEX1.GMS
gams restart file =
```

Note when a parameter is unused it is left blank. The parameter names are all listed in the GAMS Command Line Parameters chapter.

© 2022 Prof. Bruce McCarl

14.2.2.2.1.7 Write position controls

Three types of controls can be used to determine where writing is done in the file. The symbol @ controls the column number while / skips to a new line and # goes to a specified line number. The file **putex1.gms** illustrates the use of @ and /

```
put 'Run on ' system.date '  using source file  ' system.ifile ///;
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
  put @41;
  loop(scenarios,put scenarios.tl:10);
  put /;
  loop(source$sum(scenarios,
        abs(savtransport(Source,Destinaton,scenarios))),
     put 'Incoming From ' source.tl @35;
     loop(scenarios,
       put savtransport(Source,Destinaton,scenarios):10:0);
     put /;
      );
  put 'Quantity demanded ' @35
```

Notes on each follow.

Skip to a specified column: @
Skip to a new line: /
Skip to a specified row: #

When GAMS encounters a @ in a put statement, the writing position is moved to that column of the output file whether it be forward or backward from the current point. Thus, one could do one of the following **putex4.gms**

```
Put 'Hello' @3 'Goodbye';
```

which would cause an output line as follows

```
HeGoodbye
```

while

```
Put 'Hello' @20 'Goodbye';
```

yields

```
Hello               Goodbye
```

© 2022 Prof. Bruce McCarl

- Note the example above using @3 shows how one can go back and overwrite earlier text.
- One can use variables or expressions instead of fixed column numbers as in the code below from **putex4.gms**.

```
scalar width /15/;
Put 'Hello' @(width+3) 'Goodbye';
```

- @ is commonly used to align columns in the face of unequal set element widths.

When GAMS encounters a / in a put statement, the writing position is moved to the next line and placed in column one. The /'s in **putex1.gms** skip to the next line or skip several lines depending on the number of /'s used.

```
put 'Run on ' system.date '  using source file  ' system.ifile ///;
loop(Destinaton,
  put 'Report for ' , Destinaton.tl:15
  put @40 '------------  Scenario  ------------' /;
  put @41;
  loop(scenarios,put scenarios.tl:10);
  put /;
  loop(source$sum(scenarios,
        abs(savtransport(Source,Destinaton,scenarios))),
     put 'Incoming From ' source.tl @35;
     loop(scenarios,
       put savtransport(Source,Destinaton,scenarios):10:0);
     put /;
      );
  put 'Quantity demanded ' @35
```

Note GAMS does not skip to a new line unless / is used. Thus, in **putex4.gms** the line

```
loop(I, put i.tl);
```

places all the set element names on one line continuously.

```
i1        i2        i3        i4        i5        i6
```

But

```
loop(I, put i.tl /);
```

places one element per line due to use of the /.

```
i1
i2
```

© 2022 Prof. Bruce McCarl

```
i3
i4
i5
i6
```

When GAMS encounters a # in a put statement the writing position is moved to that row of the output file whether it be down or up from the current line. The file **putex4.gms** shows an example

```
Put #3 'Hello' #2 'Goodbye' #1 'Hey these are reversed';
```

which would yield

```
          Hey these are reversed
    Goodbye
Hello
```

Notes:

- The above uses of #1 shows how one can go back and write earlier lines.
- One can use variables or expressions instead of fixed column numbers as in the code below from **putex4.gms**.

```
scalar lineonpage /15/;
Put 'Hello' #(lineonpage+3) 'Goodbye';
```

- The line number is related to the current page.
- When obeying a # GAMS does not reset the column in which printing occurs as illustrated by output above. One would need to add @1 if also desiring to start in column 1.

A number of put file attributes are available to determine and or set the current position on a page where items are written or reset the last line.

.Cc
.Cr
.Hdcc
.Hdcr
.Hdll
.Ll
.Lp
.Tlcc
.Tlll
.Tlcr
.Ws

Returns or sets the current write position column in main the window. This attribute is addressed as

```
Localfileidentifier.cc= value;
```

© 2022 Prof. Bruce McCarl

or

```
parameteritem = Localfileidentifier.cc;
```

where the value can be between 1 and the page width.

Note:

The .cc suffix is updated at the conclusion of a put statement. Consequently, the .cc value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

Returns or sets current write position row in main window. This attribute is addressed as

```
Localfileidentifier.cr= value;
```

or

```
parameteritem = Localfileidentifier.cr;
```

where the value can be between 1 and the page size minus any header, title, and margins.

Note:

The .cr suffix is updated at the conclusion of a put statement. Consequently, the .cr value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

Returns or sets current write position column in the page header. This attribute is addressed as

```
Localfileidentifier.hdcc= value;
```

or

```
parameteritem = Localfileidentifier. hdcc;
```

where the value can be between 1 and the page width.

Note:

The .hdcc suffix is updated at the conclusion of a puthd statement. Consequently, the .hdcc value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

Returns or sets current write position row in the page header. This attribute is addressed as

```
Localfileidentifier.hdcr= value;
```

or

```
parameteritem = Localfileidentifier. hdcr;
```

© 2022 Prof. Bruce McCarl

where the value can be between 1 and the header size in terms of number of lines.

Note:

The .hdcr suffix is updated at the conclusion of a puthd statement. Consequently, the .hdcr value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

Returns the number of or resets the last row written in the page header. This attribute is addressed as

```
Localfileidentifier.hdll= value;
```

or

```
parameteritem = Localfileidentifier. hdll;
```

where the value can be between 1 and the header size in terms of number of lines.

Notes:

- Unlike the row and column control, the last line attribute is updated continuously.
- The .hdll attribute does not have values applicable to the current page. It will apply to the next page.
- Not only can this attribute be used to determine the last line used in a header area, but it can also be used to delete lines within this area. Namely, the header section will be completely deleted by resetting the hdll attribute to 0 and any non zero specification shorter than the current number of lines in the header deletes lines in excess of the hdll setting.

Returns the number of or resets the last row written in the page main window. This attribute is addressed as

```
Localfileidentifier.ll= value;
```

or

```
parameteritem = Localfileidentifier.ll;
```

where the value can be between 1 and the page height in terms of number of lines less adjustments for margins, headers and titles.

Notes:

- Unlike the row and column control, the Ll attribute is updated continuously.
- The .Ll attribute may be reset for the current page.
- Not only can this attribute be used to determine the last line used in a page, but it can also be used to delete lines. Namely, the current writing will be completely deleted by resetting the Ll attribute to 0 and any non zero specification shorter than the current number of lines in the header deletes lines whose line numbers are in excess of the Ll setting.

© 2022 Prof. Bruce McCarl

Put file attribute indicating the number of pages that are already in the put file. This attribute is addressed as

```
parameteritem = Localfileidentifier.lp;
```

Note that setting this to zero does not erase the pages that have previously been written to the file.

Returns or sets current write position column coordinate in the page title.

This attribute is addressed as

```
Localfileidentifier.tlcc= value;
```

or

```
parameteritem = Localfileidentifier. tlcc;
```

where the value can be between 1 and the title size.

Note:

The .tlcc suffix is updated at the conclusion of a puttl statement. Consequently, the .tlcc value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

Returns the number of or resets the last row written in the page title. This attribute is addressed as

```
Localfileidentifier.tlll= value;
```

or

```
parameteritem = Localfileidentifier. tlll;
```

where the value can be between 1 and the header size in terms of number of lines.

Notes:

- The .tlll attribute is updated continuously.
- The .tlll attribute may not be reset and have values applicable to the current page because when the title block is modified, it corresponds to the title block on the next page.
- Not only can this attribute be used to determine the last line used in a title area, but it can also be used to delete lines within this area. Namely, the title section will be completely deleted by resetting the .tlll attribute to 0 and any non zero specification shorter than the current number of lines in the title deletes lines whose numbers are in excess of the .tlll setting.

Returns or sets current write position row coordinate in the page title.

This attribute is addressed as

© 2022 Prof. Bruce McCarl

```
Localfileidentifier.tlcr= value;
```

or

```
parameteritem = Localfileidentifier. tlcr;
```

where the value can be between 1 and the title size in terms of number of lines.

Note:

The .tlcr suffix is updated at the conclusion of a puttl statement. Consequently, the .tlcr value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

The .ws attribute allows the user to determine the number of rows that can be written to the main window on the current page. This attribute is addressed as

```
parameteritem = Localfileidentifier.ws;
```

This attribute shows the number of rows that can be placed on the page, 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 attribute is calculated by GAMS and is not changeable by the user.

14.2.2.2.2 Formatting of items

Lines and pages can be formatted on a local or global basis. This formatting permits control of field width, decimals, and item justification. A number of put page attributes can also be altered including put page length (lines per page), page with, page footer format, and page header format. Finally, the upper/ lower case characteristics of the font used in the printout can be controlled.

14.2.2.2.2.1 File formatting – append or overwrite

The put writing facility has the ability to append to or overwrite an existing file. The put file attribute .ap determines which occurs.

The syntax for the .ap attribute is

```
Localfileidentifier.ap= value;
```

Where a value of

0 causes the put file to overwrite (replace) the existing
1 causes the information generated by the puts to be appended to the file.

```
My1.ap = 1;
```

Notes:

- Any items put into the active put file from that point on will be added to the end of the existing file.

© 2022 Prof. Bruce McCarl

- If the file does not exist, it will be created.

14.2.2.2.2.2 Page formatting

Pages can be formatted in terms of height and width using put file attributes. 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

.Bm - bottom margin
.Lm - left margin
.Pc - Page control
.Ps or page height
.Pw - page width
.Tm - top margin

This gives the number of blank lines to be placed in the bottom margin of the page. These lines add lines to the page height beyond the number of lines specified in the .ps command. This is set using the syntax

```
Localfileidentifier.bm=number;
```

or in the example **putex5.gms**

```
my1.bm=13;
```

This only works when the page control (pc) option is 0.

This gives the number of blank columns to be placed on the left of the page. This is set using the syntax

```
Localfileidentifier.lm=number;
```

or in the example **putex5.gms**

```
my1.lm=13;
```

An important attribute that controls how pages appear, as well as whether a number of the other parameters (.ps, .tm, .bm) even function and whether the data are printed in a comma delimited fashion is the page control option. This is set using the syntax

```
Localfileidentifier.pc=number;
```

or in the example **putex5.gms**

```
my1.pc=3;
```

© 2022 Prof. Bruce McCarl

The integer 0-6 are allowable values for number.

Value	Resultant effect on Output file
0	Causes the use of 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	Causes the use of Fortran page format. This option places the numeral one in the first column of the first row of each page in following standard Fortran convention.
2	Causes no paging to be done and is the default setting.
3	Causes ASCII page control characters to be inserted.
4	Makes the put file output into a space delimited file. Non-numeric output is quoted, and each item is delimited with a blank space. Here # and @ commands are ignored.
5	Makes the put file output into a comma delimited (CSV) file. Non-numeric output is quoted, and each item is delimited with a comma. Here # and @ commands are ignored.
6	Makes the put file output into a tab delimited file. Non-numeric output is quoted, and each item is delimited with a tab. Here # and @ commands are ignored.

The last three options create delimited files, and are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets. See the example **putex6.gms**.

This gives the height of a page in terms of the number of rows (lines) that can be placed on any page of the document. This is set using the syntax

```
Localfileidentifier.ps= number;
```

or in the example **putex5.gms**

```
my1.ps=65;
```

Notes:

- By default GAMS does not try to control pagination of put files making no allowance for paging. This works well when the user is going to import the file into a word processor and is willing to let the word processor do the paging.
- The page size commands **will be ineffective** unless a non default value is used for the page control (pc) setting. In the example we use

  ```
  my1.pc=3;
  ```

- When earlier pages have been printed out which are shorter than the current page size specified GAMS generates an error message.
- Maximum page size is 130 lines and the minimum is 60 lines.

© 2022 Prof. Bruce McCarl

This gives the width of a put file page in terms of the number of columns that can be placed on a line. This is set using the syntax

```
Localfileidentifier.pw=number;
```

or in the example **putex5.gms**

```
my1.pw=81;
```

Notes:

- GAMS by default allows a 255 character wide page.
- When the material printed out exceeds the current page width specified GAMS generates an error message and places 4 asterisks (****) in the last 4 columns of the line.
- Maximum page width is effectively unlimited with values up to 32767 allowable.

Number of blank lines to be placed at the top margin of the page. These lines add lines to the page height above and beyond the number of lines specified in the .ps command. Default value is 0. This is set using the syntax

```
Localfileidentifier.tm=number;
```

or in the example **putex5.gms**

```
my1.tm=3;
```

This works for any of the page control (pc) settings.

14.2.2.2.2.3 Inserting Tabs

When one wishes to include tabs in the put files within the command system.tab can be used as follows

```
file test / test.put /;
put test; put "1","%system.tab%","2";
```

Where the syntax **"%system.tab%"** results in a tab character being put into the file.

14.2.2.2.2.4 Adding page titles and headers

Pages may be formatted with titles and headers. There are actually three independent writing areas on each page of a document. These areas are the title block, the header block, and the main window. The primary purpose for these independent writing areas is to allow the user to create a file where the title and possibly the header is repeated across multiple pages.

© 2022 Prof. Bruce McCarl

The layout of the page is

Title block
Header block
Main window

The Put command writes to the Main window, and there are special commands Puttl and Puthd that write to the title and header. By default the title and header are empty. The title and header blocks are essentially the same as the main window and use exactly the same syntax rules. These are discussed below.

Puttl places information in the title block and is formatted as is the normal put command with the syntax (**putex5.gms**)

```
Puttl Item;
```

Example:

```
Puttl "File written on" system.date /
```

Notes:

- Once a Puttl command is executed the items therein are placed in the title block.
- The title block is displayed on each subsequent page unless modified.
- The heading block can be emptied out using Putclear.
- Once the main window for a page has been written to, any further modifications of the title block will be shown on subsequent pages and not the current page.
- Every page must have an entry in the main window. 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 ' ';
```

- The size of any area within a given page is based entirely on the number of lines put into it.
- 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.
- Paging occurs automatically whenever a page is full.
- Each area of a page is maintained independently, so we can write with a Puttl for a while then a Puthd, then back to a Puttl. However once we use Put all subsequent Puttl and Puthd go to the next and subsequent pages, not the current page.

© 2022 Prof. Bruce McCarl

Puthd places information in the heading block and is formatted as is the normal put command with the syntax

```
Puthd Item;
```

Example:

(putex5.gms)

```
Puthd "Page " system.page /;
```

Notes:

- Once a Puthd command is executed the items therein are placed in the heading block.
- The heading block is displayed on each subsequent page unless modified.
- The heading block can be emptied out using Putclear.
- Once the main window for a page has been is written to, any further modifications of the header block will be shown on subsequent pages and not the current page.
- Every page must have an entry in the main window. When a page has no output in its main 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 ";
- The size of any area within a given page is based entirely on the number of lines put into it.
- 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.
- Paging occurs automatically whenever a page is full.
- Each area of a page is maintained independently, so we can write with a Puttl for a while then a Puthd, then back to a Puttl. However once we use Put all subsequent Puttl and Puthd go to the next and subsequent pages, not the current page.

Putclear removes the contents of the heading and title blocks and is formatted with the syntax

```
Putclear Item;
```

An example appears in **putex5.gms**.

14.2.2.2.2.5 Upper low er font case formatting: .Case and .Lcase

These attributes are used to specify the case in which alphabetic characters are displayed in the output file. The syntax employed is

```
Localfileidentifier.case=number;
```

© 2022 Prof. Bruce McCarl

There are 3 allowable values for number.

Value	Resultant effect on the put file
0	causes mixed case to be displayed.
1	causes the output to be displayed in upper case regardless of the case used for the input.
2	causes the output to be displayed in lower case regardless of the case used for the input.

The Lcase attribute does casing for set element names using the syntax

```
Localfileidentifier.lcase=number;
```

There are 3 allowable values for number.

Value	Resultant effect on set elements names in the put file
0	Causes mixed case to be displayed.
1	Causes lower case to be displayed.
2	Causes upper case to be displayed.

14.2.2.2.2.6 Width and decimal formatting

Users may desire to control the spacing and decimal format of output items. For formatting purposes, there are four categories of output items: labels, numeric values, set values, and quoted/explanatory text. Within GAMS a global width and decimal place default format is assumed for each category and then a local item dependent choice can be made within a specific put statement.

Both field width and, in the case of numerical data, the number of decimal places may be controlled on a global basis. The GAMS defaults for these items follow:

Item	Default value	Symbol
Set element name field width	12	.lw
Numeric field width	12	.nw
Numeric decimal Places (maximum 10)	2	.nd
Set element entry (yes and no) width	12	.sw
Quoted and Explanatory text field width	0	.tw

The general syntax for specifying these formatting items is

```
Localfileidentifier.symbol=value;
```

© 2022 Prof. Bruce McCarl

The value for the width specified is the exact number of characters that will be employed. However a specification of 0 causes the field width to be the minimum size that will fully display the item.

Special actions are taken when the item does not fit. If an item containing text contains more characters than the field width, the text will be printed starting from the left with characters beyond the width omitted. For items containing numeric values, the decimal portion of a number is rounded or scientific notation will be used to fit the number within the given field and if that wont work a set of *'s is entered.

The way of overriding each of these defaults involves use of an attribute of the localfileidentifier put file name. The attributes that can be used are the symbols listed in the last column of the table above. Use of each will be discussed below.

This attribute controls the width of set element names in the put output. It is addressed using the syntax

```
Localfileidentifier.lw=number;
```

or in the example **putex6.gms**

```
my1.lw=12;
```

Example:

Using the sequence (**putex6.gms**)

```
set mine abcdefghijklmnopqrstuvwxyz
/a12345678901234567890  setaabcdefghijklmnopqrstuvwxyz
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small smallone/;
loop(mine,
put 'start set element text here $' mine.tl '$ end here'/;)
put /;
```

where the items in orange are the text for set element names and the blue items are statements causing those to be output into the put file. In turn with the default for **.lw of 12** we get

```
start set element name here $a12345678901$ end here
start set element name here $b12345678901$ end here
start set element name here $small       $ end here
```

where short text entries like small are padded with trailing blanks but the long names like a12345678901234567890 are truncated to their first 12 positions a12345678901 compared to the full set element name specified just above.

If we reset **lw to 20** we get

```
start set element name here $a12345678901234567890          $ end here
start set element name here $b12345678901234567890          $ end here
start set element name here $small                          $ end here
```

© 2022 Prof. Bruce McCarl

It we reset **lw to 0** we get

```
start set element name here $a123456789012345678900$ end here
start set element name here $b123456789012345678900$ end here
start set element name here $small$ end here
```

showing the exact width and full contents.

This attribute controls the decimals in numerical items in the put output. It is addressed using the syntax

```
Localfileidentifier.nd=number;
```

or in the example **putex6.gms**

```
my1.nw=2;
```

Example:

Using the example **putex6.gms**

```
scalar number regnumber /1.2356/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start number here       $':0 number '$ end here'/;
put 'start small number here $':0 smallnumber '$ end here'/;
put 'start large number here $':0 largenumber '$ end here'/;
```

In turn when we run this with the default for **.nd of 2** we get

```
start number here       $        1.24$ end here
start small number here $        0.00$ end here
start large number here $1.0000000E+9$ end here
```

showing that by default 2 decimals are printed, unless the number is too large and is moved into exponential format. It also shows numbers are rounded. Note small numbers can also be suppressed with nz.

If we reset **nd to 0** we get

```
start number here       $           1$ end here
start small number here $           0$ end here
start large number here $  1000000000$ end here
```

This attribute controls the width of numerical items in the put output. It is addressed using the syntax

```
Localfileidentifier.nw=number;
```

or in the example **putex6.gms**

© 2022 Prof. Bruce McCarl

```
my1.nw=12;
```

Example:

Using the example **putex6.gms**

```
scalar number regnumber /1.2356/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start number here       $':0 number '$ end here'/;
put 'start small number here $':0 smallnumber '$ end here'/;
put 'start large number here $':0 largenumber '$ end here'/;
```

where the items in orange are the numerical data and the blue items are statements causing those to be output into the put file. In turn when we run this with the default for **.nw of 12** we get

```
start number here       $        1.24$ end here
start small number here $        0.00$ end here
start large number here $1.0000000E+9$ end here
```

showing that by default 12 characters are always printed, that the number is padded to the left, and reported in exponential format if too large.

If we reset **nw to 0** we get

```
start number here       $1.24$ end here
start small number here $0.00$ end here
start large number here $1000000000.00$ end here
```

showing the exact width is used with the specified decimals as discussed above.

If we reset **nw to 4** we get

```
start number here       $1.24$ end here
start small number here $0.00$ end here
start large number here $****$ end here
```

showing that when a number is still too large, asterisks replace the value in the output file.

This attribute controls the width printout for the set element entries that are YES and NO. It is addressed using the syntax

```
Localfileidentifier.sw=number;
```

or in the example **putex6.gms**

```
my1.sw=12;
```

© 2022 Prof. Bruce McCarl

Example:

Using the example **putex6.gms**

```
set mine abcdefghijklmnopqrstuvwxyz
/a12345678901234567890  setaabcdefghijklmnopqrstuvwxyz
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small smallone/;
set small(mine) /small/;
loop(mine,
put 'start set element here $':0 mine(mine) '$ end here for name ' mine.tl /;)
put /;
loop(mine,
put 'start subset element here $':0 small(mine) '$ end here for name ' mine.tl /
put /;
```

where the items in orange are the sets for which we will put out element indicators and the blue items are statements causing those to be output into the put file. In turn, when we run this with the default for **.sw of 12** we get

```
start set element value here $         YES$ end here for name a12345678901
start set element value here $         YES$ end here for name b12345678901
start set element value here $         YES$ end here for name small

start subset element value here $          NO$ end here for name a12345678901
start subset element value here $          NO$ end here for name b12345678901
start subset element value here $         YES$ end here for name small
```

showing that by default a left padded 12 characters are always printed. We could also reset to tw to 20 or 0 with the same effect as discussed for lw or tw above.

This attribute controls the width of explanatory text items, and set element names plus quoted text in the put output. It is addressed using the syntax

```
Localfileidentifier.tw=number;
```

or in the example **putex6.gms**

```
my1.tw=12;
```

Example:

Using the example **putex6.gms**

```
set mine abcdefghijklmnopqrstuvwxyz
/a12345678901234567890  setaabcdefghijklmnopqrstuvwxyz
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small smallone/;
set small(mine) /small/;
```

© 2022 Prof. Bruce McCarl

```
scalar number regnumber /1.23456/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start quoted text here $':0 'Quotedabcedeghijklmnopqrstuvwxyz'
             '$ end here'//;
put 'start item explanatory text here $':0 mine.ts '$ end here'/;
put 'start item explanatory text here $':0 number.ts '$ end here'/;
put /;
loop(mine,
put 'start set element explanatory text here $':0 mine.te(mine) '$ end here'/;)
put /;
```

where the items in orange are explanatory or quoted text and the blue items are statements causing those to be output into the put file. In turn, when we run this with the default for .tw of 0 we get

```
start quoted text here $Quotedabcedeghijklmnopqrstuvwxyz$ end here

start item explanatory text here $abcdefghijklmnopqrstuvwxyz$ end here
start item explanatory text here $regnumber$ end here

start set element explanatory text here $setaabcdefghijklmnopqrstuvwxyz$ end her
start set element explanatory text here $setbabcdefghijklmnopqrstuvwxyz$ end her
start set element explanatory text here $smallone$ end here
```

showing that by default the full length is always printed.

If we reset **tw to 20** we get

```
start quoted text here $Quotedabcedeghijklmn$ end here

start item explanatory text here $abcdefghijklmnopqrst$ end here
start item explanatory text here $regnumber           $ end here

start set element explanatory text here $setaabcdefghijklmnop$ end here
start set element explanatory text here $setbabcdefghijklmnop$ end here
start set element explanatory text here $smallone            $ end here
```

where short text entries like smallone are padded with trailing blanks but the long names like **Quotedabcedeghijklmnopqrstuvwxyz** are truncated to their first 20 positions **Quotedabcedeghijklmn** compared to the output just above.

While global formatting is nice sometimes certain items require individual attention. GAMS provides item specific formatting, which overrides global format settings. For text items the syntax of this feature is as follows:

```
Put item:width;
```

While for numeric items it is

```
Put item:width:decimals;
```

© 2022 Prof. Bruce McCarl

The item, width, and decimals are delimited with colons as shown above. The width is length in characters and has all the characteristics for sets, numbers and text as discussed above under the global section. A zero width again causes use of the exact width. The decimals feature behaves as discussed under the global section above. Examples from **putex7.gms** follow

```
set mine abcdefghijklmnopqrstuvwxyz
/a12345678901234567890  setaabcdefghijklmnopqrstuvwxyz
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small smallone/;
set small(mine) /small/;
scalar number regnumber /1.2356/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start quoted text here $':0 'Quotedabcedeghijklmnopqrstuvwxyz':5
             '$ end here'//;
put 'start item explanatory text here $':4 mine.ts:22 '$ end here'/;
put 'start item explanatory text here $':0 number.ts:7 '$ end here'/;
put /;
loop(mine,
put 'start set element name here $'0 mine.tl:0 '$ end here'/;)
put /;
loop(mine,
put 'start set element explanatory text here $':0 mine.te(mine):0 '$ end here'/
put /;
loop(mine,
put 'start set element value here $':0 mine(mine):5 '$ end here for name ' mine
put /;
loop(mine,
put 'start subset element value here $':0 small(mine) :0 '$ end here for name '
put /;
put 'start number here       $':0 number:10:2 '$ end here'/;
put 'start number here       $':0 number:10:4 '$ end here'/;
put 'start number here       $':0 number:15:4 '$ end here'/;
put 'start number here       $':0 number:0:4 '$ end here'/;
```

and the result is

```
start quoted text here $Quote$ end here

starabcdefghijklmnopqrstuv$ end here
start item explanatory text here $regnumb$ end here

start set element naa12345678901234567890$ end here
start set element nab12345678901234567890$ end here
start set element nasmall$ end here

start set element explanatory text here $setaabcdefghijklmnopqrstuvwxyz$ end her
```

© 2022 Prof. Bruce McCarl

```
start set element explanatory text here $setbabcdefghijklmnopqrstuvwxyz$ end her
start set element explanatory text here $smallone$ end here

start set element value here $  YES$ end here for name a12345678901
start set element value here $  YES$ end here for name b12345678901
start set element value here $  YES$ end here for name small

start subset element value here $NO$ end here for name a12345678901
start subset element value here $NO$ end here for name b12345678901
start subset element value here $YES$ end here for name small

start number here       $      1.24$ end here
start number here       $    1.2356$ end here
start number here       $         1.2356$ end here
start number here       $1.2356$ end here
```

showing width and decimal adjusted labels and numbers. The zero width entries cause exact fits as in the last line.

It is worthwhile reiterating that a width choice of zero allows one to mix GAMS output in without extra spacing as illustrated by **putex8.gms**

```
set i /i1 Nuts
       i2 Bolts/;
parameter jdata Hardware Data
              /i1 22.73
               i2 100.918/;
put 'Here is a report for the ' jdata.ts:0 ' entries in my GAMS code' //;
loop(i,
    put @5 'For element ' i.tl:0 ' named ' i.te(i):0 ' the data are '
          jdata(i) :0:3 ' as entered' /;
     );
```

where the red shows zero with specifications for text data and numerical data. In turn we get

```
Here is a report for the Hardware Data entries in my GAMS code
    For element i1 named Nuts the data are 22.730 as entered
    For element i2 named Bolts the data are 100.918 as entered
```

14.2.2.2.2.7 Justification

Users may desire to control justification of output items so they are centered, aligned left or aligned right. Justification may be controlled for four categories of output items: labels, numeric values, set values, and quoted text. Within put files, a global default justification format is assumed and then a local item dependent justification choice can be made within a specific put statement.

On a global basis GAMS specifies defaults for the 4 categories of put file items as follows

© 2022 Prof. Bruce McCarl

Item	Default Justification	Justification Value	Symbol
Set element name	Left	2	.lj
Numeric output	Right	1	.nj
Set element entry (yes's and no's)	Right	1	.sj
Quoted and Explanatory text	Left	2	.tj

This can be changed using the format

```
Localfileidentifier.justificationsymbol=value;
```

The value specified is

1 for justification to the right of a field
2 for justification to the left of a field
3 for justification in the center of a field

Notes:

- Justification only occurs when the width of the field in which an element is to be placed exceeds the width of the element to be put into that field.
- The justificationsymbol for overriding each of these defaults is an attribute of the localfileidentifier put file name and is the symbol in the last column of the table above. Use of each will be discussed below.

This attribute controls the width of set element names in the put output. It is addressed using the syntax

```
Localfileidentifier.lj=number;
```

or in the example **putex9.gms**

```
my1.lj=3;
```

where 1 is for right, 2 for left and 3 for center.

Example:

Using the sequence (**putex6.gms**)

```
set mine abcdefghijklmnopqrstuvwxyz
/a1  seta1
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small/;
```

© 2022 Prof. Bruce McCarl

```
loop(mine,
put 'start set element name here $':0 mine.tl:20 '$ end here'/;)
put /;
```

where the items in orange are set element text and the blue items are statements causing those to be output into the put file. In turn, with the default for **.lj of 2** we get

```
start set element name here $a1                    $ end here
start set element name here $b12345678901234567890$ end here
start set element name here $small                 $ end here
```

where narrow text entries like a1 and small are flushed left, but the long names like a12345678901234567890 just fill the field.

If we reset **lj to 1** we get

```
start set element name here $                    a1$ end here
start set element name here $b12345678901234567890$ end here
start set element name here $                 small$ end here
```

where we see the narrow entries flushed right. If we reset lw to 3 we get

```
start set element name here $          a1          $ end here
start set element name here $b12345678901234567890$ end here
start set element name here $       small          $ end here
```

showing centering of the narrow entries.

This attribute controls the justification of numerical items in the put output. It is addressed using the syntax

```
Localfileidentifier.nj=number;
```

or in the example **putex6.gms**

```
my1.nw=12;
```

where 1 is for right, 2 for left and 3 for center.

Example:

Using the example **putex9.gms**

```
scalar number regnumber /1.2356/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start number here       $':0 number:20:4 '$ end here'/;
put 'start small number here $':0 smallnumber:20:4 '$ end here'/;
put 'start large number here $':0 largenumber:20:4 '$ end here'/;
```

© 2022 Prof. Bruce McCarl

In turn, when we run this with the default for **.nj of 1** we get

```
start number here       $              1.2356$ end here
start small number here $              0.0000$ end here
start large number here $     1000000000.0000$ end here
start large number here $1.00000E+9$ end here
```

showing the default flush right for narrow items. If we reset **nj to 2** we get

```
start number here       $1.2356              $ end here
start small number here $0.0000              $ end here
start large number here $1000000000.0000     $ end here
start large number here $1.00000E+9$ end here
```

showing the left flush for narrow items. If we reset **nj to 3** we get

```
start number here       $       1.2356       $ end here
start small number here $       0.0000       $ end here
start large number here $  1000000000.0000   $ end here
start large number here $1.00000E+9$ end here
```

showing centering for narrow items.

This attribute controls the put file justification for the set element entries that are YES and NO. It is addressed using the syntax

```
Localfileidentifier.sj=number;
```

or in the example **putex9.gms**

```
my1.sj=2;
```

where 1 is for right, 2 for left and 3 for center.

Example:

Using the example **putex9.gms**

```
set mine abcdefghijklmnopqrstuvwxyz
/a1  seta1
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small/;
set small(mine) smallone /small/;
loop(mine,
put 'start set element here $':0 mine(mine) '$ end here for name ' a /;)
put /;
loop(mine,
put 'start subset element here $':0 small(mine) '$ end here for name ' mine.tl /
put /;
```

© 2022 Prof. Bruce McCarl

where the items in orange are the sets for which we will put out element indicators and the blue items are statements causing those to be output into the put file. In turn when we run this with the default for **.sj of 1** we get

```
start set element value here $       YES$ end here for name a1
start set element value here $       YES$ end here for name b12345678901
start set element value here $       YES$ end here for name small

start subset element value here $        NO$ end here for name a1
start subset element value here $        NO$ end here for name b12345678901
start subset element value here $       YES$ end here for name small
```

showing the default flush right. If we reset **sj to 2** we get

```
start set element value here $YES       $ end here for name a1
start set element value here $YES       $ end here for name b12345678901
start set element value here $YES       $ end here for name small

start subset element value here $NO        $ end here for name a1
start subset element value here $NO        $ end here for name b12345678901
start subset element value here $YES       $ end here for name small
```

showing the left flush. If we reset **sj to 3** we get

```
start set element value here $   YES  $ end here for name a1
start set element value here $   YES  $ end here for name b12345678901
start set element value here $   YES  $ end here for name small

start subset element value here $    NO    $ end here for name a1
start subset element value here $    NO    $ end here for name b12345678901
start subset element value here $    YES   $ end here for name small
```

showing centering.

This attribute controls the justification of explanatory text for items, and set element names plus quoted text in the put output. It is addressed using the syntax

```
Localfileidentifier.tj=number;
```

or in the example **putex9.gms**

```
my1.tj=2;
```

where 1 is for right, 2 for left and 3 for center.

Example:

Using the example **putex9.gms**

© 2022 Prof. Bruce McCarl

```
set mine abcdefghijklmnopqrstuvwxyz
/a1  seta1
 b12345678901234567890  setbabcdefghijklmnopqrstuvwxyz
 small/;
set small(mine) smallone /small/;
scalar number regnumber /1.2356/
put 'start quoted text here $':0 'Quot':15 '$ end here'//;
put 'start quoted text here $':0 'Long Quoted Text Entry':15
     '$ end here'//;
put 'start item explanatory text here $':0 mine.ts:20
     '$ end here'/;
put 'start item explanatory text here $':0 number.ts:20 '
     $ end here'/;
loop(mine,
put 'start set element name here $':0 mine.tl:20 '$ end here'/;)
put /;
```

where the items in orange explanatory or quoted text and the blue items are statements causing those to be output into the put file. In turn when we run this with the default for **.tj of 2** we get

```
start quoted text here $Quot           $ end here
start quoted text here $Long Quoted Tex$ end here
start item explanatory text here $abcdefghijklmnopqrst$ end here
start item explanatory text here $regnumber           $ end here
start set element explanatory text here $seta1               $ end here
start set element explanatory text here $setbabcdefghijklmnop$ end here
start set element explanatory text here $small               $ end here
```

showing the default flush left for narrow items.

If we reset **tj to 1** we get

```
start quoted text here $           Quot$ end here
start quoted text here $Long Quoted Tex$ end here
start item explanatory text here $abcdefghijklmnopqrst$ end here
start item explanatory text here $           regnumber$ end here
start set element explanatory text here $               seta1$ end here
start set element explanatory text here $setbabcdefghijklmnop$ end here
start set element explanatory text here $               small$ end here
```

showing the right flush for narrow items. If we reset **tj to 3** we get

```
start quoted text here $     Quot      $ end here
start quoted text here $Long Quoted Tex$ end here
start item explanatory text here $abcdefghijklmnopqrst$ end here
start item explanatory text here $     regnumber      $ end here
start set element explanatory text here $       seta1        $ end here
start set element explanatory text here $setbabcdefghijklmnop$ end here
start set element explanatory text here $       small        $ end here
```

© 2022 Prof. Bruce McCarl

showing centering for narrow items.

While global formatting is nice sometimes certain items require individual attention. GAMS provides item specific formatting for this, which overrides global format settings. For text items the syntax of this feature is as follows:

```
Put item:justificationsymbol width;
```

While for numeric items it is

```
Put item:justificationsymbol width:decimals;
```

The justificationsymbol is

>	for right flush
<	for left flush
<>	for centering

and the item, width, decimals and colons are as laid out above.

Examples:

(putex10.gms)

```
file my1;
put my1;
set mine abc
/a1  seta1
 b1234  setbabhijklmnopqrstuvwxyz
 small/;
set small(mine) smallone /small/;
scalar number regnumber /1.2356/
       smallnumber /0.00000001/
       largenumber /1000000000/;
put 'start quoted text here $':0 'Quot':>15 '$ end here'/;
put 'start quoted text here $':0 'Quot':<15 '$ end here'/;
put 'start quoted text here $':0 'Quot': <>15 '$ end here'//;
put 'start item explanatory text here $':0 mine.ts: >20 '$ end here'/;
put 'start item explanatory text here $':0 mine.ts: <20 '$ end here'/;
put 'start item explanatory text here $':0 mine.ts: <>20 '$ end here'//;
put /;
loop(mine,
put 'start set element name here $':0 mine.tl: >20 '$ end here'/;)
put /;
```

© 2022 Prof. Bruce McCarl

```
loop(mine,
put 'start set element explanatory text here $':0 mine.te(mine):<20 '$ end here
put /;
loop(mine,
put 'start set element value here $':0 mine(mine): <>10
     '$ end here for name ' mine.tl /;)
put /;
put 'start number here       $':0 number: <20:4 '$ end here'/;
put 'start number here       $':0 number: >20:4 '$ end here'/;
put 'start number here       $':0 number:<>20:4 '$ end here'/;
put 'start large number here $':0 largenumber:10:4 '$ end here'/;
put 'start large number here $':0 largenumber: <10:4 '$ end here'/;
put /;
```

and the result is

```
start quoted text here $          Quot$ end here
start quoted text here $Quot           $ end here
start quoted text here $     Quot      $ end here
start item explanatory text here $               abc$ end here
start item explanatory text here $abc               $ end here
start item explanatory text here $        abc       $ end here
start set element name here $                 a1$ end here
start set element name here $               b1234$ end here
start set element name here $               small$ end here
start set element explanatory text here $seta1              $ end here
start set element explanatory text here $setbabhijklmnopqrstu$ end here
start set element explanatory text here $small              $ end here
start set element value here $   YES    $ end here for name a1
start set element value here $   YES    $ end here for name b1234
start set element value here $   YES    $ end here for name small
start number here       $1.2356             $ end here
start number here       $              1.2356$ end here
start number here       $       1.2356       $ end here
start large number here $1.00000E+9$ end here
start large number here $1.00000E+9$ end here
```

where items are flushed right by the use of > , others are flushed left by the use of < and some are centered by the use of <> . Note the green entries are unaffected since the item fills up the field.

14.2.2.2.2.8 Additional numeric display control

In addition to the field width and justification controls discussed in the previous sections, the following put file attributes can be globally specified for numeric display.

This is a scientific format toggle. It's use allows one to vary the contents of the puts with respect to the formatting of numeric values in terms of use of scientific notation (E format) as opposed to GAMS choice or fixed decimal place format (F). It is addressed using the syntax

© 2022 Prof. Bruce McCarl

```
Localfileidentifier.nr=number;
```

The potential entries for number and the result are

0 entries displayed in F or E format
1 numbers rounded to fit fields
2 all numbers displayed in scientific notation (E format)
3 numbers with floating decimal and constant precision

Gives the tolerance level below which numbers with smaller absolute values are treated as zero. When this is set equal to zero, rounding is determined by the field width. The default value for the attribute is 1.0e-5. It is addressed using the syntax

```
Localfileidentifier.nz=realnumber;
```

Where realnumber is the value below which any numerical output with smaller absolute values will be reported as zero.

14.2.2.2.3 Putclose

The Putclose keyword is used to close a file during the execution of program. Otherwise GAMS automatically closes files when it exits.

Example:

One application where this is useful involves writing an option file that controls a solver from within a GAMS model. In that case, the file must be closed prior to the SOLVE statement and Putclose allows that to be done. The following example shows the creation and closing of an option file for the MINOS solver as implemented in **frstpart.gms**.

```
FILE  OPT   MINOS option file  / MINOS5.OPT /;
Put OPT;
Put 'BEGIN'/
    '  Iteration limit        500'/
    '  Major damping parameter 0.5'/
    '  Feasibility tolerance   1.0E-7'/
    '  Scale all variables'/
    'END';
Putclose OPT;
```

One may also wish to report what loop one is on in a large model using commands like (**comparw.gms**)

```
$set console
$if %system.filesys% == UNIX  $set console /dev/tty
$if %system.filesys% == DOS $set console con
$if %system.filesys% == MS95  $set console con
$if %system.filesys% == MSNT  $set console con
$if "%console%." == "." abort "filesys not recognized";
```

© 2022 Prof. Bruce McCarl

```
file screen / '%console%' /;
```

followed by

```
loop(scenarios,
    put screen;
    put 'I am on scenario ' Scenarios.tl;
    putclose;
```

that writes to the screen (or DOS window that must be made visible in the IDE using file options execute). In this case Putclose is necessary to cause the line to become visible. One can also use Putclose to change the title of the DOS box.

Notes:

- This program segment would be placed inside the GAMS model prior to the solve statement.
- If the internal file name is omitted from the Putclose statement, the current Put file is closed.
- 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 you normally would. The existing file will either be overwritten or appended to depending on the value of the append file (.ap) suffix.

14.2.2.2.4 Putpage

GAMS does automatic paging when the lines printed reaches the page height (ps) is reached. Through the use of the command Putpage a page can be terminated and written to the file when the user desires. Putpage forces the current page to be immediately written, making a new page available for Put statements. Putpage can be used with output items. When it is, the page is written including the output items contained in the Putpage statement.

```
Putpage OUT 'This text is placed in the window and the page ended';
```

14.2.3 Putting out a block of text: $onput, $offput, $onputs, $onputv

Sometimes one needs to put out several lines of text just for appearances sake. In such a case using conventional puts one would have syntax like the following (**putex12.gms**)

```
Put 'Line 1 of text' /;
Put 'Line 2 of text' //;
Put 'Line 3 of text' /;
Put 'Line 4 of text' /;
```

One may use the commands $onput and $offput as an alternative

```
$onput
Line 1 of text
Line 2 of text

Line 3 of text
Line 4 of text
```

© 2022 Prof. Bruce McCarl

```
$offput
```

Yielding idenetical put files. In this case the text block is begun with $onput and ended with $offput.

There is also a variant, 2, that uses substitutable parameters (Parameter substitution is discussed in the Conditional Compilation chapter). Namely $onputs will cause all parameter statements in it (those enclosed in % signs) to be replaced with any defined GAMS calling parameters of control variables. $onputv suppresses any substitution. For example the sequence (**putex12.gms**)

```
$setglobal it "from $onputs"
$onputs
substitution
Line 1 of text "%it%"
Line 2 of text %it%
Line 3 of text %it%
Line 4 of text %it%
$offput
```

Generates the output

```
substitution
Line 1 of text "from $onputs"
Line 2 of text from $onputs
Line 3 of text from $onputs
Line 4 of text from $onputs
```

While (**putex12.gms**)

```
$setglobal it "from $onputs"
$onputv
No substitution
Line 1 of text "%it%"
Line 2 of text %it%
Line 3 of text %it%
Line 4 of text %it%
$offput
```

Generates

```
No substitution
Line 1 of text "%it%"
Line 2 of text %it%
Line 3 of text %it%
Line 4 of text %it%
```

14.2.4 Making puts conditional

As with other GAMS statements, conditionals can be used with Put statements to control whether

© 2022 Prof. Bruce McCarl

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';
```

14.2.5 Output to other programs

There are cases where one wishes to save things to other programs. This is generally done using Put files. Again one can write customized programs or can use some of Rutherford's tools. Two examples of the customized puts follow.

Put of data to a regression code
Put file for export to mapping program

14.2.5.1 Put of data to a regression code

(regput.gms)

Here we put data for use in a program like SPSS. We use puts to print out fixed formatted files.

```
file tosass;
put tosass;
loop(run,
  put run.tl;
  put @12;
put /;
loop(decwant,s=  fawelsum( "Agconswelf",decwant,run)/1000;put s:13:0;);
put /;
loop(decwant,s=  fawelsum( "AGGOVTCOST",decwant,run)/1000;put s:13:2;);
put /;
loop(decwant,s= agtable(decwant,"agtradbal",run)/1000;put s:13:2;);
  put / ;
 ) put / ;
```

Sample of Data saved

```
r1
             30           40           50
          60.00        70.00        80.00
           7.56        15.11        22.67
r2
             40           50           60
          70.00        80.00        90.00
           7.56        15.11        22.67
r3
             50           60           70
          80.00        90.00       100.00
           7.56        15.11        22.67
```

© 2022 Prof. Bruce McCarl

Each set of lines gives the case name for one model run followed by the data points.

14.2.5.2 Put file for export to mapping program

(maplink.gms)

Here we put a csv-delimited file for export to a mapping program.

```
sets meas /nitrogen,phosporous,potassium,cropland,
                  watererosn,winderosn,sediment,pub-water,pumpwater,
                  chemicalco/;
sets region /EAST,STHEAST,MIDWEST,WEST,STHCENTRAL,NORTHERNPL  /
table data(region,meas) data to be put
               nitrogen   phosporous  potassium  chemicalco  cropland
EAST              0.96      -0.17       0.52      -0.24       0.00
STHEAST           0.13       0.09       0.13      -0.12       0.02
MIDWEST           0.40       0.36       0.54      -0.03       0.36
WEST              1.74       1.51       1.73       0.59       1.63
STHCENTRAL       -0.09      -0.15       0.04       0.17       0.12
NORTHERNPL        3.55       1.70       2.59       3.16       1.65
  +            watererosn winderosn sediment pub-water pumpwater
EAST             -1.14      0.01     -1.16      0.00    -10.71
STHEAST           3.13      0.67      3.64      0.34      0.00
MIDWEST          -0.23      0.59     -0.23      0.00     -1.11
WEST              0.57      0.02      0.74      0.01      0.26
STHCENTRAL       -0.16     -1.33     -0.08      0.00     -1.55
NORTHERNPL        0.92     -3.06      0.85      0.00     -0.07

file mapdat;
put mapdat;
mapdat.pw=250;
set s1(meas) /nitrogen,phosporous,potassium,chemicalco,cropland/
put '"region"'; loop(s1,put ' , "' s1.tl '"'); put /;
loop(region,
   put '"' region.tl '"'; loop(s1,put ',' data(region,s1):10:2); put /);
set s2(meas) / watererosn,winderosn,sediment,pub-water,pumpwater/
put '"region"'; loop(s2,put ' , "' s2.tl '"'); put /;
loop(region,
   put '"' region.tl '"'; loop(s2,put ',' data(region,s2):10:2); put /);
```

the resultant output is

```
"region","nitrogen","phosporous","potassium","cropland","chemicalco"
"EAST",0.96,-0.17,0.52,0.00,-0.24
"STHEAST",0.13,0.09,0.13,0.02,-0.12
"MIDWEST",0.40,0.36,0.54,0.36,-0.03
"WEST",1.74,1.51,1.73,1.63,0.59
"STHCENTRAL",-0.09,-0.15,0.04,0.12,0.17
```

© 2022 Prof. Bruce McCarl

```
"NORTHERNPL",3.55,1.70,2.59,1.65,3.16
"region","watererosn","winderosn","sediment","pub-water","pumpwater"
"EAST",-1.14,0.01,-1.16,0.00,-10.71
"STHEAST",3.13,0.67,3.64,0.34,0.00
"MIDWEST",-0.23,0.59,-0.23,0.00,-1.11
"WEST",0.57,0.02,0.74,0.01,0.26
"STHCENTRAL",-0.16,-1.33,-0.08,0.00,-1.55
"NORTHERNPL",0.92,-3.06,0.85,0.00,-0.07
```

This is also readily importable to a spreadsheet.

14.2.6 Errors that arise during puts

Errors can occur during execution of put commands that are caused when 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, their presence is sometimes overlooked without reviewing the program listing and these put errors might go undetected, especially in large output files. Consequently, GAMS includes a file attribute to help one detect errors:

```
localfileidentifier.errors
```

This attribute allows one to display the number of put errors occurring during execution of put commands to a file. The attribute is addressed using the syntax

```
Parametervalue = Localfileidentifier.errors;
```

Such statements can be inserted at any point of a program to detect the number of errors, which have occurred up to that location.

14.3 Acronyms

In GAMS an **acronym** is a special data type that allows the use of character strings as values.

Declaration
Usage

14.3.1 Declaration

Acronyms are defined in a fashion similar to set or parameter definitions. The basic format is

```
Acronym Itemname Explanatory text;
```

or

© 2022 Prof. Bruce McCarl

```
Acronyms Itemname Explanatory text;
```

Where Itemname names the acronym and follows the same rules discussed in the Rules for Item Names, Element names and Explanatory Text chapter. Explanatory text is associated text that is used for self-documentation. The Explanatory text entry again must follow the rules given in the Rules for Item Names, Element names and Explanatory Text chapter.

Examples:

```
acronym ALABEL Here is a label
acronyms monday, tuesday, wednesday, thursday, friday ;
```

Notes:

- Acronym or acronyms can be used interchangeably.
- The name of the acronym is also the text it takes on when displayed.

14.3.2 Usage

Acronyms can be used in scalar, parameter and table statements as well as in calculation - assignment statements and conditionals.

Examples:

(acronym.gms)

```
acronyms nameforit,nextone;
acronym  acronym3  the third one
acronym  doit any old acronym
set i /i1, i2, i3 /;
parameter textstrings(i)
     /i1 nameforit,     i2 nextone,      i3 acronym3/ ;
parameter textstring(i)
     /i1 doit,      i2 1,      i3 doit/ ;
display textstrings;
file putacronym;
put putacronym;
put nameforit //
loop(i,if(textstrings(i)= nameforit, put 'Something');
put textstrings(i) / );
scalar flagtome;
flagtome=doit;
equation aa(i);
variables x(i);
aa(i)$(textstring(i)=doit).. 3*x(i)=e=1;
display flagtome,textstring;
parameter zz(i),rr(i);
zz(i)=textstring(i)  ;
display zz;
```

© 2022 Prof. Bruce McCarl

```
table acks(i,i)
              i1              i2              i3
i1                                           doit
i2                            doit
i3           doit;
display acks;
```

Notes:

- Scalar, parameter, or table statements are entered as described in the Data Entry chapter with acronym names used wherever numbers could be entered.
- Conditionals can be used, but when dealing with acronyms you can only use eq or ne operators not gt or lt.
- When acronyms are included in Assignment statements the statements can only set other items equal to acronyms or other parameters containing acronyms.
- Numerical operations (+ - * / **) cannot be done with acronyms.
- Numbers can be mixed in with acronyms in a parameter that is defined over a set or sets, but once the acronym is in the parameter then that parameter cannot be subsequently manipulated numerically.

14.4 Conditional Compilation

GAMS provides commands that allow significant compile time changes in the basic structure of any set of GAMS instructions dependent on user defined items. These features allow one do several things including

- Simplify maintenance of models that share common features but have significant differences involving features that could not simultaneously exist in a compiled GAMS code.
- Add memory, execution time or solver requirements that are desirable to avoid if not needed.
- Develop utilities that may be used across a wide variety of applications in different contexts.

Any of these items can be structured so that they can be activated or deactivated by the choice of a single controlling variable, as I will show below.

Control variables
Environment variables
$If and $Ifi conditionals
Forms of conditionals
Incorporating Goo: $Goto and $Label
Redefining expressions
Running external programs or commands
Writing messages to LST, LOG and other files
End the job: $Exit, $Abort, $Error, $Stop, $Terminate
Longer examples

© 2022 Prof. Bruce McCarl

14.4.1 Control variables

Control variables can be used to cause compile time changes in program structure. They can be defined, listed in terms of current status and used in $If conditionals.

Establishing control variables

14.4.1.1 Establishing control variables

Control variables are defined in three different ways.

```
$set varname value
$setlocal varname value
$Setglobal varname value
--varname=value
```

where

varname is any user chosen variable name

value is optional in all cases but the last one and can contain text or a number

The main difference between these definitions involves the accessibility of the control variables within code brought into a program through the family of include commands.

These variables can be destroyed using

```
$Drop varname
$Droplocal varname
$Dropglobal varname

where varname is as above.
```

14.4.1.1.1 $Setglobal

Control variables can be set that are available throughout the code.

These are set using syntax like (control.gms)

```
$setglobal gg what
$setglobal heregg
--mycommand=value
```

where

gg and heregg are the names of the two control variables being defined what is the text being assigned to the control variable gg

the -- syntax is explained here

If one wishes to remove the control variables they are destroyed using

© 2022 Prof. Bruce McCarl

```
$dropglobal varname
```

Unfortunately 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 as discussed here.

14.4.1.1.2 $Setlocal

Control variables can be set that are available only in the code module where the are defined using $setlocal defined.

These are set using syntax like (control.gms)

```
$setlocal yy no
$setlocal hereyy
```

where

yy and hereyy are the names of the two local control variables being defined

no is the text being assigned to the control variable yy

If one wishes to remove the control variables they are destroyed using

```
$droplocal varname
```

Unfortunately GAMS allows one to define local and global variables with the same name but treats them as different under some cases and prioritizes them when using $ife or $if as discussed here.

14.4.1.1.3 $Set

Control variables can be set which are called scoped and are available in the code module where defined and any code included therein, but not anywhere in code that includes the module where the $Set command appears.

These are set using syntax like (control.gms)

```
$set it 1
$set hereit
```

where

it and hereit are the names of the two scoped control variables being defined

1 is the text being assigned to the control variable it

If one wishes to remove the control variables they are destroyed using

. $drop varname

© 2022 Prof. Bruce McCarl

The -- syntax also defines a scoped variable but it is in the main module and is thus equivalent to a global variable.

14.4.1.1.4 $EvalGlobal

This evaluates a numerical expression at compile time and places it into a global control variable. In turn one can use $ife to do numeric testing on the value of this variable.

The format is

$EvalGlobal expression

The expression must consist of constants, functions or other control variables with numerical values.

Example:

```
$set znumber 4
$Evalglobal anumber %znumber%+10
$ife %anumber%>14 display "it exceeds 14" "%anumber%"
$ife %anumber%<14 display "it is less than 14" "%anumber%"
$ife %anumber%=14 display "it equals 14" "%anumber%"
$ife %anumber%<>14 display "it does not equal 14" "%anumber%"
```

Notes:

- Related functions are

$Eval	that works with scoped control variables
$Evallocal	that works with local control variables

- Unfortunately 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 as discussed here.
- The calculations can only involve real numbers
- Expression evaluation proceeds from left to right
- When items are not set off in parentheses the operator precedence is:

```
OR XOR EQV IMP
AND
NOT
< <= = <> >= > LE LE EQ NE GE GT
+ - binary and unary
* /
^ **
```

- The following functions can be used in expressions:

```
abs  ceil  cos  exp   floor  frac  IfThen  log   log2  log10  max
min  mod   PI   power round  sign  sin     sleep sqr   sqrt   tan
```

© 2022 Prof. Bruce McCarl

```
trunk
```

14.4.1.1.5 $Evallocal

This evaluates a numerical expression at compile time and places it into a local control variable. In turn one can use $ife to do numeric testing on the value of this variable.

The format is

$EvalLocal expression

The expression must consist of constants, functions or other control variables with numerical values.

Example:

```
$set znumber 4
$EvalLocal anumber %znumber%+10
$ife %anumber%>14 display "it exceeds 14" "%anumber%"
$ife %anumber%<14 display "it is less than 14" "%anumber%"
$ife %anumber%=14 display "it equals 14" "%anumber%"
$ife %anumber%<>14 display "it does not equal 14" "%anumber%"
```

Notes:

- Related functions are

  $Eval that works with local control variables
  $Evalglobal that works with global control variables

- Unfortunately 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 as discussed here.
- The calculations can only involve real numbers
- Expression evaluation proceeds from left to right
- When items are not set off in parentheses the operator precedence us

```
OR XOR EQV IMP
AND
NOT
< <= = <> >= > LE LE EQ NE GE GT
+ - binary and unary
* /
^ **
```

- The following functions can be used in expressions:

```
abs  ceil  cos  exp  floor  frac  IfThen  log  log2  log10  max
```

© 2022 Prof. Bruce McCarl

```
min  mod   PI   power round  sign  sin     sleep sqr   sqrt   tan
 trunk
```

14.4.1.1.6 $Eval

This evaluates a numerical expression at compile time and places it into a scoped control variable. In turn one can use $ife to do numeric testing on the value of this variable.

The format is

$Eval expression

The expression must consist of constants, functions or other control variables with numerical values. The following GAMS date and release functions may also be used: jdate, jtime, gyear, gmonth, gday, gdow, gleap, ghour, gminute, gsecond, gmillisec, jstart, jnow, gamsversion, and gamsrelease

Example:

```
$set znumber 4
$Eval anumber %znumber%+10
$ife %anumber%>14 display "it exceeds 14" "%anumber%"
$ife %anumber%<14 display "it is less than 14" "%anumber%"
$ife %anumber%=14 display "it equals 14" "%anumber%"
$ife %anumber%<>14 display "it does not equal 14" "%anumber%"
```

Notes:

- Related functions are

  $EvalLocal that works with local control variables
  $EvalGlobal that works with global control variables
- Unfortunately 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 as discussed here.
- The calculations can only involve real numbers
- Expression evaluation proceeds from left to right
- When items are not set off in parentheses the operator precedence is:

```
OR XOR EQV IMP
AND
NOT
< <= = <> >= > LE LE EQ NE GE GT
+ - binary and unary
* /
^ **
```

© 2022 Prof. Bruce McCarl

- The following functions can be used in expressions:

  abs ceil cos errorlevel exp fact floor frac gamsrelease gamsversion gday gdow ghour gleap gmillisec gminute gmonth gsecond gyear IfThen jdate jnow jobhandle jobkill jobstatus jobterminate jstart jtime log log2 log10 max min mod PI power round sign sin sleep sqr sqrt tan trunc

14.4.1.2 Setting environment variables

GAMS recognizes the environment variable GDXCONVERT and GDXCOMPRESS which control the format with which GDX files are written.

They are set using the syntax

$setenv GDXCOMPRESS number

$setenv GDXCONVERT V#

with the number and v# allowable values as discussed in the Gdxcompress and Gdxconvert parameters sections.

14.4.1.3 Destroying Contol Variables

These variables can be destroyed using

```
$Drop varname
$Droplocal varname
$Dropglobal varname
```

where varname is the control variable name and the .local etc. corresponds to the nature of the control vari

14.4.1.4 A problem with control variable definitions

Unfortunately 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.

Namely it appears as if the local setting is always used first then the scoped then the global. An example appears in setcontrol.gms

Note

In the face of this it may be advisable to only use one type of control variable in any application or at least only use a name once.

© 2022 Prof. Bruce McCarl

14.4.2 Environment variables

Windows NT contains a set of named system and user defined environment variables. A technical discussion of such items appears at *https://support.microsoft.com/?scid=kb;en-us;*. GAMS allows one to specify conditionals in response to environment variable values, set user environment variables to particular values and destroy user environment variables.

Names of some system environment variables
Defining and destroying user environment variables
Augmenting environment variables
Accessing environment variable status at any point in the code: $Show

14.4.2.1 Names of some system environment variables

The system environment variables contain among other things the ones listed below. A longer list can be found at *https://web.archive.org/web/20081104192846/http://kennethhunt.com/archives/000933.html*.

CD	%CD%	current directory string.
COMPUTERNAME	%COMPUTERNAME%	name of the computer.
DATE	%DATE%	current date.
ERRORLEVEL	%ERRORLEVEL%	error code of the most recently used command.
HOMEPATH	%HOMEPATH%	path of user's home directory. .
NUMBER_OF_PROCESSORS	%NUMBER_OF_PROCESSORS%	number of processors installed on the computer.
OS	%OS%	OS name. Windows XP and Windows 2000 display as Windows_NT.
PATH	%PATH%	System specifies the search path for executable files.
TIME	%TIME%	current time

Note of caution: The following standard environment variables are not always available:

© 2022 Prof. Bruce McCarl

CD

CMDCMDLINE

CMDEXTVERSION

DATE

ERRORLEVEL

PROMPT

RANDOM

TIME

14.4.2.2 Defining and destroying user environment variables

Environment variables are defined as follows.

```
$setenv varname value
```

where

varname is a user chosen environment variable name

value can contain text or a number

Environment variables are destroyed as follows.

```
$dropenv varname
```

where varname is a user chosen environment variable name.

14.4.2.3 Augmenting environment variables

Environment variables may have items added to their beginning or their end as follows.

To add to the beginning

```
$setenv varname %sysenv.varname% value
```

To add to the beginning

```
$setenv varname value %sysenv.varname%
```

where

© 2022 Prof. Bruce McCarl

varname is a user chosen environment variable name

value contains text or a number

%sysenv.varname% is the old text that was in the environment variable.

Note of caution using $SETENV Under windows 2000, XP and vista, environment variables are defined locally. After the GAMS model finished execution, the setting of the environment variable is lost. $SETENV can therefore not be used to pass information from a GAMS model to the outside world outside the confines of the GAMS model itself. Similarly only environment variables defined at core windows level are available. Even environment variables that were created by spawning an external program from GAMS are not available.

Example:

```
$onecho > cmd.cmd

set xxx=test

set xxx > cmd.set

$offecho

$call cmd.cmd


$echo xxx=%sysenv.xxx% > cmd.xxx
```

The result in cmd.set and cmd.xxx are not identical!!!!

14.4.2.4 Accessing environment variable status at any point in the code: $Show

One may look at environment variable values by displaying them or by creating a simple program that displays them using an echo command and then using a $call to invoke it.

Examples:

(environvar.gms)

```
* Display current values of the system environment variables like the PATH varia
display '%sysenv.PATH%'

*display environment varialbe in log file

*first set up a simple command processor
$onechoV > xx.cmd
echo  the path is %path%
echo  the os is %os%
$offecho
```

© 2022 Prof. Bruce McCarl

```
*now run it
$call xx.cmd
```

14.4.3 $If and $Ifi conditionals

One can employ conditional statements using the $If, $Ifi and $Ife possibly with the Not modifier.

$If and $Ifi
$Ife
Not as a modifier

14.4.3.1 $If and $Ifi

The $If statement causes execution of a GAMS statement when a conditional is **true**. The basic form of the $If and $Ifi command is

```
$If conditional statementtoexecute
$Ifi conditional statementtoexecute
```

or with the instruction on the next line

```
$If conditional
     statementtoexecute
```

or

```
$Ifi conditional
     statementtoexecute
```

The net effect is that the code that GAMS will execute will either contain or not contain statementtoexecute depending on the true false nature of the conditional. In particular, when the conditional in the $If or $Ifi test is true, then the statementtoexecute will be entered into the source code to be executed. If not, then the statements statementtoexecute will never be executed.

Notes:

- The $If makes comparisons involving text in a case sensitive fashion. The $Ifi is a case insensitive variant.
- The conditional is evaluated at compile time, so does not involve GAMS calculated numbers.
- An example is in **basicif.gms**.
- Surrounding a control variable name with % signs substitutes the text for that control variable in that place as illustrated and discussed below.

14.4.3.2 $Ife conditionals

This evaluates numerically the conditional at compile time.

© 2022 Prof. Bruce McCarl

```
$Ife conditional statementtoexecute
```

If the conditional is true, then, the GAMS statement, i.e. statementtoexecute, is added to the GAMS program at compile time. The conditional can be:

```
item
```

or

```
item == item2
```

Let's first consider the first alternative:

```
$Ife item statementtoexecute
```

The item may be a term or expression. A term may be a control variable value, defined scalar or constant. An expression consists of:

```
<term><op><term>
```

An expression does not allow spaces. Furthermore, we can use functions on a term. The operator (op) may be one of the following:

= to test equality
> to test greater than
< to test less than
<> to test not equal

Several expressions can be combined by nesting operators, e.g. `and` and `or`. Brackets can be used to separate expressions. Instead of separating the expression with brackets, we can also surround the sequence of expressions with quotes or double quotes and separate the expressions with spaces. Consider the following example:

```
$set   a   2
scalar b / 3 /;
$ife (%a%=2)and(b=3)and(sqr(b)=9) $log Test 1 returns true
$ife "%a%=2 and b=3 and sqr(b)=9" $log Test 2 returns true
```

Test 1 and 2 returns true. Note that, we evaluate a control variable by surrounding it with a percent sign, e.g. the value of control variable a is referenced by `%a%`.

Let us now consider the second alternative:

```
$Ife item == item2 statementtoexecute
```

For a conditional with numerical evaluation, $ife, the special operator == tests for equality with a tolerance of 1E-12, however, for a conditional, $if, it denotes a string comparison. In a $ife statement the special operator == can only be used ones and spaces are allowed on both sides of

© 2022 Prof. Bruce McCarl

the special operator. Consider the following example:

```
$ife 0 == 1E-12  $log Test 3 returns true
$ife 1==1+1E-13  $log Test 4 returns true
```

Test 3 returns false and test 4 returns true.

Observe that the control variable may obtain its value from a evaluating an expression, see eval, valglobal or evallocal .

Example: (setcontrol.gms)

```
$set znumber 4
$Evalglobal anumber %znumber%+10
$Evalglobal vnumber %znumber%+9

$ife %anumber%>14 display "it exceeds 14" ,"%anumber%" ;
$ife %anumber%<14 display "it is less than 14" ,"%anumber%"  ;
$ife %anumber%=14 display "it equals 14" ,"%anumber%";
$ife %anumber%<>14 display "it does not equal 14", "%anumber%";
$ife %anumber%>%vnumber% display "it exceeds vnumber", "%anumber%", "%vnumber%
$ife %anumber% display "anumber is nonzero";
$ife %anumber%<>0 display "anumber is nonzero";
```

Notes:

- Unfortunately 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 as discussed here.
- When testing whether item == item2 the test is true if abs (item2-item)/ (1+abs(item2)) < 10^(-12)
- When the command is used without an operator as in

  $ife %anumber% display "anumber is nonzero";

  then it is same as

  $ife %anumber% <> 0 display "anumber is nonzero";

  i.e. the program tests whether an item is nonzero.
- The evaluation of expressions follows the rules given under the discussion of $Eval

14.4.3.3 Not as a modifier

The $If statement can contain a Not modifier. If so the GAMS statementtoexecute will be compiled and executed only when the conditional is **false**. The basic form of the $If not command is

```
$If NOT conditional statementtoexecute
$Ifi NOT conditional statementtoexecute
```

© 2022 Prof. Bruce McCarl

Example:

(basicif.gms)

```
scalar x /1/;
*$ondollar
scalar y /1/;
$Setglobal gg
$Setglobal tt doit
$If setglobal gg display x;
$If not setglobal gg display y;
$If "%tt%" == "doit" x=x*2;
$Ifi "%tt%" == "DOIT" x=y**2;
$If "%tt%" == "DOIT" x=x*100;
$If not "%tt%" == "doit" y=y/4;
$set aa yes
$If "%aa%" == yes y=x;
```

The resultant effect on the compiled code is

```
1  scalar x /1/;
2  *$ondollar
3  scalar y /1/;
6  display x;
8  x=x*2;
9  x=y**2;
13  y=x;
```

Note lines in blue are suppressed because the $If fails while the magenta, orange, red and violet lines occur since the conditionals are true. The contrast between the lines

```
$Ifi "%tt%" == "DOIT" x=y**2;
$If "%tt%" == "DOIT" x=x*100;
```

where the second line fails since tt was set to "doit". This shows the case sensitive nature of $If versus the case insensitivity of $Ifi.

14.4.4 $ifthen, iftheni, ifthene, else, elseif, endif conditionals

$Ifthen and the other components below are a form of a $IF that controls whether a number of statements are active. An $IFTHEN must be matched with a $ENDIF. The syntax for the condition are generally the same as for the $if statement. The **$ifthen** and **$elseif** have variants that are case insensitive ($IFi and $ELSEIFi) or evaluate numerical values of the control variables ($IFe and $ELSEIFe).

Notes:

- $IFTHENe is used to do numerical comparisons

© 2022 Prof. Bruce McCarl

- $IFTHENi is used to do case insensitive comparisons, while $IFTHEN does case sensitive ones
- $ELSEIF has another comparison behind it as in the example below
- $ELSEIFi is a case insensitive variant of $Eleseif
- $ELSEIFe is a numerical value evaluating variant of $Eleseif
- The statements following directly a **$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 statements.
- A NOT maybe used in the commands
- The comparisons allowed are covered in the forms of conditionals section
- One may add a tag to the IFTHEN and ENDIF conditions to force them the match up such as in **(setcontrol.gms)**

```
$ifthen.onea x == x
      display "it";
$ifthen.twoa a == a
display "it2";
$endif.twoa
$endif.onea
```

- The evaluation of expressions follows the rules given under the discussion of $Eval

Examples:

(setcontrol.gms)

```
$setglobal aroundit
$ifthen setglobal aroundit
        display "statement 1";
        display "statement 2";
$else
        display "statment 3";
$endif
```

Here when aroundit is setglobal then we get the two displays executed. Otherwise the third one occurs.

Much more complex forms can be used

```
$maxgoto 10 $set x a
$label two
```

© 2022 Prof. Bruce McCarl

```
$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

$eval x 1
$label three
display 'x=%x%';
$ifthen %x% == 1 $eval x %x%+1
$elseif %x% == 2 $eval x %x%+1
$elseif %x% == 3 $eval x %x%+1
$elseif %x% == 4 $eval x %x%+1
$else            $set  x done
$endif $if NOT %x% == done $goto three
```

Lengthy and nested ithen/else structures can become difficult to debug. Tagging of the begin, the $ifthen and the end, the $endif can be helpful. For example, the next line will fail because the tags do not match:

```
$ifthen.one x == x
$endif.one
```

As with the $if statement, the statement on the line with the **$ifthen** style statements is optional. The following two statements give the same results:

```
$iftheni %type% == low $include abc
$elseifi %type% == med $include efg
$else                  $include xyz
$endif

$iftheni %type% == low
$include abc
$elseifi %type% == med
$include efg
$else
$include xyz
$endif
```

The statements following directly a $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 statements.

```
$ifthen.two   c==c  display 'true for tag two';
$ifthen.three a==a  $log true for tag three
display '   then clause for tag three';
$ifthen.four x==x display 'true for tag four';
$log true for tag four
$else
```

© 2022 Prof. Bruce McCarl

```
display '      else clause for tag four';
$endif.four          $log endif four
$endif.three         $log endif three
$endif.two           $log endif two
```

This will produce a GAMS program like

```
1  display 'true for tag two';
3  display '   then clause for tag three';
4  display 'true for tag four';
```

with the following log output

```
--- Starting compilation
true for tag three
true for tag four
endif four
endif three
endif two
```

14.4.5 Forms of conditionals

The conditionals used in $If and $Ifi statements can be of many forms. The principal ones involve:

- Comparisons involving control variables or environment variables testing whether a variable is defined or whether its contents match a string.
- Characteristics of a named item testing its type, set dependency or declaration status.
- GAMS command line parameters including user defined options.
- The settings of system characteristics.
- Batinclude parameters testing existence, contents, declaration status, or type.
- Error checks.
- File existence status.

Each is discussed below.

Based on control and environment variables
Based on characteristics of named item or parameter
Passed parameter existence
Based on GAMS command line parameters
Based on system characteristics
Based on error and warning checks
Based on file existence
Based on put file status

© 2022 Prof. Bruce McCarl

14.4.5.1 Based on control and environment variables

When one inserts conditionals in a GAMS program based on control variables, one either tests for the existence of a control variable or does tests on the contents of the control variable.

14.4.5.1.1 Existence

One can use a $If or $Ifi with or without the not modifier to test for whether or not the control variable has been set and if so execute a command otherwise skipping that command. The test to see if a control variable has been set is of one of three forms:

```
$If set       controlvariablename statementtoexecute;
$If setglobal controlvariablename statementtoexecute;
$If setlocal  controlvariablename statementtoexecute;
$If setenv    envronmentvariablename statementtoexecute;
```

The evaluation of each as to whether it is true or false depends on the characteristics of the set command used in its original definition. Several cases arise:

- If the variable identified by controlvariablename or envronmentvariablename was set by any of the $set, $setenv, $setlocal, or $Setglobal commands it will render the $If involving the plain set conditional to be true.
- If the control variable was established using $Setglobal, then any conditional involving the setglobal test will also be true.
- If the control variable was established using $setlocal, then any conditional using the setlocal condition would become true.
- If the control variable was not established with any of these statements it will not pass any of the conditionals.
- $If or $Ifi can be used interchangeably with the not modifier included if desired.

Example:

(basicif.gms)

```
$setlocal  alocal yes
$set       aset yes
$Setglobal aglobal yes
acronyms local,global,plainset
scalar type;
*set item
$If set       aset   type=plainset;
$If setglobal aset   type=global;
$If setlocal  aset   type=local;
*setglobal
$If set       aglobal type=plainset;
$If setglobal aglobal type=global;
$If setlocal  aglobal type=local;
*setlocal
$Ifi set       alocal type=plainset;
```

© 2022 Prof. Bruce McCarl

```
$Ifi setglobal alocal type=global;
$Ifi setlocal  alocal type=local;
*not set
$If set       qq type=plainset;
$If setglobal qq type=global;
$If setlocal  qq type=local;

$If not set       qq display "No qq around";
```

results in the echo print

```
17  *set item
18  type=plainset;
21  *setglobal
22  type=plainset;
23  type=global;
25  *setlocal
26  type=plainset;
28 type=local;
29  *not set
34  display "No qq around";
```

where line 18 shows that the command variable established with the $set the syntax passes only the $If set conditional while the setglobal command variable passes both the $If set and the $If setglobal conditionals. Also $setlocal command variable passes both the $Ifi set and the $Ifi setlocal condition. Finally, the referenced command variable qq that was never established within a set command did not pass any of the conditions except the last one with the not in it.

Examples employing the environmental variables appear in **environvar.gms**.

14.4.5.1.2 Contents

One can test the contents of a control variable by using several syntax forms below

```
$If %controlvariablename% == texttocompare     statementtoexecute;
$If %controlvariablename%"  == "texttocompare" statementtoexecute;
$If "%controlvariablename%" == texttocompare    statementtoexecute;
$If "%system.environmentvarname%" == "texttocompare" statementtoexecute;
```

Notes:

- The %controlvariablename% retrieves the text that was placed in the control variable. The % precedes and succeeds the control variable name. The quotes need to be used when spaces were included in the text placed therein.
- The texttocompare is a text string to compare the text in the control variable with.
- The == is a symbol saying compare these two strings.
- The % symbol may be redefined using the Escape dollar command.

© 2022 Prof. Bruce McCarl

Examples:

(basicif.gms)

```
*texttocompare
$Setglobal controlvariablename       comparethistext
$If       %controlvariablename%  == "comparethistext" type1=ok;
$If       %controlvariablename%  ==  comparethistext  type2=ok;
$If      "%controlvariablename%" == "comparethistext" type3=ok;
$If      "%controlvariablename%" ==  comparethistext  type4=ok;

*add spaces
$Setglobal controlvariablename       compare this text
$If      "%controlvariablename%" == "compare this text" type1=ok;
$If      "%controlvariablename%" ==  compare this text  type2=ok;
```

generates code that after compilation and looks like the following

```
39  *texttocompare
41  type1=ok;
42  type2=ok;
43  type3=ok;
44  type4=ok;
45
46  *add spaces
47  type1=ok;
```

This code shows that the quoted or uncontrolled quoted control variable name and target text works when spaces are not present. If spaces are present then the text retrieved by use of %controlvariablename% must be encased in quotes and if the comparison is to work so must be the target text.

Often one may wish to see if the text for an item is blank. This is frequently done with commands like the following

```
$If       "%controlvariablename%a"  == "a"  display "blank it is";
$If not "%controlvariablename%a" == "a" display "it is not blank";
```

which contain the element "%controlvariablename%a" that reduces just to "a" if the named control variable does not have a text entry.

Examples employing the environmental variables appear in **environvar.gms**.

14.4.5.1.3 Numerical Value

One can test the numerical value of a control variable by using several syntax forms below

```
$Ife %controlvariablename% op valuetocompare     statementtoexecute;
$Ife %controlvariablename%  statementtoexecute;
```

© 2022 Prof. Bruce McCarl

Notes:

- $IFe or $IFTHENe must be used to do numerical comparisons
- The %controlvariablename% retrieves the value that was placed in the control variable. The % precedes and succeeds the control variable name. .
- The valuetocompare is a numerical value to compare with and can be another control variable name enclosed in % signs
- The op is a symbol saying compare these two values.and is

=	to test equality
==	to test equality with a tolerance
>	to test greater than
<	to test less than
<>	to test not equal

- When == is used the test is when testing whether item1 == item2 the test is true if abs (item2-item1)/ (1+abs(item2)) < 10^(-12)
- When just the control variable name is used then the test is whether or not it holds a non zero value

Examples:

(setcontrol.gms)

```
$set znumber 4
$Evalglobal anumber %znumber%+10
$Evalglobal xnumber %znumber%+9
$ife %anumber%>14 display "it exceeds 14", "%anumber%";
$ife %anumber% == %anumber2% Display "they are close" ;
$ife %anumber%<14 display "it is less than 14", "%anumber%"
$ife %anumber%=14 display "it equals 14", "%anumber%"
$ife %anumber%<>14 display "it does not equal 14", "%anumber%"
$ife %anumber%>%xnumber% display "it exceeds xnumber","%anumber%","%xnumber%"
$ife %anumber% display "anumber is nonzero again";
```

generates code that after compilation and looks like the following

```
39  *texttocompare
41  type1=ok;
42  type2=ok;
43  type3=ok;
44  type4=ok;
45
46  *add spaces
47  type1=ok;
```

This code shows that the quoted or uncontrolled quoted control variable name and target text works when

© 2022 Prof. Bruce McCarl

spaces are not present. If spaces are present then the text retrieved by use of %controlvariablename% must be encased in quotes and if the comparison is to work so must be the target text.

Often one may wish to see if the text for an item is blank. This is frequently done with commands like the following

```
$If       "%controlvariablename%a"  == "a"  display "blank it is";
$If not "%controlvariablename%a" == "a" display "it is not blank";
```

which contain the element "%controlvariablename%a" that reduces just to "a" if the named control variable does not have a text entry.

Examples employing the environmental variables appear in **environvar.gms**.

14.4.5.2 Based on characteristics of named item or parameter

One can use a $If or $Ifi with or without the not modifier to test the characteristics of a named item or a passed parameter in one of the include with arguments (batinclude, libinclude, sysinclude) commands in terms of

- Type of GAMS symbol (set, parameter, model, variable, equation, or file).
- Item definition and declaration status.
- Number of sets the item is defined with respect to.
- Existence of passed parameters.

14.4.5.2.1 Item type

A named item in a GAMS program can be one of eight fundamental types. They are set, parameter, model, variable, equation, file, acronym, or function. A conditional can be structured to test whether a passed argument or named item is of a particular type and in turn invoke type specific processing. The syntax then would be one of the following where the text in pink is not part of the statement but rather a definition

```
$If acrtype itemname gamsstatement   is this item an acronym
$If equtype itemname gamsstatement   is this item an equation
$If funtype itemname gamsstatement   is this item a GAMS function
$If modtype itemname gamsstatement   is this item a model
$If filtype itemname gamsstatement   is it a local name for put file
$If partype itemname gamsstatement   is this item a parameter
$If settype itemname gamsstatement   is this item a set
$If vartype itemname gamsstatement   is this item a variable
```

If the item identified by itemname is the type signified by the keyword with the prefix and the word type the subsequent gamsstatement would be entered into the active code and executed. If not the command is skipped.

Example:

(iftype.gms)

© 2022 Prof. Bruce McCarl

```
set itemname;
$If acrtype itemname  display "itemname is an acronym";
$If equtype itemname  display "itemname is an equation";
$If funtype itemname  display "itemname is a GAMS function";
$If modtype itemname  display "itemname is an model";
$If filtype itemname  display "itemname is localname for put file";
$If partype itemname  display "itemname is a parameter";
$If settype itemname  display "itemname is a set";
$If vartype itemname  display "itemname is a variable";
$If xxxtype itemname  display "%itemname% is a xxx";
$If pretype itemname  display "%itemname% is a pre";
$If protype itemname  display "%itemname% is a pro";
```

results in

```
 3    set itemname;
10 display "itemname is a set";
```

One also can use a batinclude version **iftypeinc.gms** as is included by **calliftypeinc.gms**

```
$If acrtype %1  display "%1 is an acronym";
$If equtype %1  display "%1 is an equation";
$If funtype %1  display "%1 is a GAMS function";
$If modtype %1  display "%1 is an model";
$If filtype %1  display "%1 is localname for put file";
$If partype %1  display "%1 is a parameter";
$If settype %1  display "%1 is a set";
$If vartype %1  display "%1 is a variable";

$If xxxtype %1  display "%1 is a xxx";
$If pretype %1  display "%1 is a pre";
$If protype %1  display "%1 is a pro"
```

along with (**iftype.gms**)

```
set aset;
acronym acro;
$batinclude iftypeinc aset
$batinclude iftypeinc acro
$batinclude iftypeinc sqrt
```

results in

```
  17  set aset;
  18  acronym acro;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\IFTYPEINC.GMS
  28  display "aset is a set";
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\IFTYPEINC.GMS
  37  display "acro is an acronym";
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\IFTYPEINC.GMS
```

© 2022 Prof. Bruce McCarl

```
54  display "sqrt is a GAMS function";
```

14.4.5.2.2 Definition status: Declared and Defined

A named item or item referenced by a passed parameter may be declared but may never have had any data put into it or may be both declared and have data. A conditional can be structured to test whether an argument or named item was ever defined or declared. The syntax then would be one of the following

```
$If declared itemname gamsstatement
```

which tests whether the item has been declared in a set, parameter, table, model, equation, file, acronym, or variable statement

```
$If defined itemname gamsstatement
```

that tests whether the item has been defined with data somewhere.

Example

(basicif.gms)

```
set aaa;
scalar bbb /1/;
$If not declared aaa display 'aaa is not declared';
$If not declared bbb display 'bbb is not declared';
$If not declared ccc display 'ccc is not declared';
$If not defined  aaa display 'aaa is not defined';
$If not defined  bbb display 'bbb is not defined';
$If not defined  ccc display 'ccc is not defined';
```

yields

```
59  set aaa;
60  scalar bbb /1/;
63  display 'ccc is not declared';
64  display 'aaa is not defined';
66  display 'ccc is not defined';
```

where aaa has no data (in this case no assigned set elements) and the $If causes generation of an action (in this case a display) since it is not being defined and the ccc tests generate both messages since it is not mentioned anywhere.

14.4.5.2.3 Set dependency: Dimension

An item may be declared as being indexed by zero through 20 sets. A conditional can be structured to determine how many sets are involved in the declaration. The syntax then would be one of the following

To test if this item of dimension zero (a scalar)

```
$If dimension 0 itemname gamsstatement
```

© 2022 Prof. Bruce McCarl

To test if this item of dimension 1 (a parameter with one index set -- a(i))

```
$If dimension 1 itemname gamsstatement
```

To test if this item of dimension 3 (a parameter a(i,j,k))

```
$If dimension 3 itemname gamsstatement
```

Note cases 0-10 are allowed.

Example:

Suppose I use this syntax within a file that I will batinclude. In that file I use conditionally compiled statements to change formatting of the option relative to the display of an item to reallocate the items in rows and columns dependent upon the number of sets that defines a passed named GAMS item. In this case the file to batinclude is **dimdisp.gms** and call it with a single argument.

```
*testing %1
$If dimension  0 %1 display '%1 is 0 dimensional',%1;
$If dimension  1 %1 display '%1 is 1 dimensional',%1;
$If dimension  2 %1 option %1:0:1:1;display '%1 is  2 dimensional',%1;
$If dimension  3 %1 option %1:0:1:2;display '%1 is  3 dimensional',%1;
$If dimension  4 %1 option %1:0:1:3;display '%1 is  4 dimensional',%1;
$If dimension  5 %1 option %1:0:1:4;display '%1 is  5 dimensional',%1;
$If dimension  6 %1 option %1:0:2:4;display '%1 is  6 dimensional',%1;
$If dimension  7 %1 option %1:0:2:5;display '%1 is  7 dimensional',%1;
$If dimension  8 %1 option %1:0:2:6;display '%1 is  8 dimensional',%1;
$If dimension  9 %1 option %1:0:2:7;display '%1 is  9 dimensional',%1;
$If dimension 10 %1 option %1:0:2:8;display '%1 is 10 dimensional',%1;
```

In turn suppose I call the file several times over with different objects are different dimensions (**basicif.gms**).

```
scalar eee /1/;
set set1 /a,b/;
set set2 /d,e/;
set set3 /1,2/;
set set4 /g,h/;
parameter fff(set1);
fff(set1)=1;
set ggg(set1,set2,set3,set4);
ggg(set1,set2,set3,set4)=yes;
parameter hhh(set1,set2,set3,set4,set4);
hhh(set1,set2,set3,set4,set4)=ord(set1)+ord(set4);
$batinclude dimdisp eee
$batinclude dimdisp set1
$batinclude dimdisp fff
$batinclude dimdisp ggg
$batinclude dimdisp hhh
```

© 2022 Prof. Bruce McCarl

The resultant code that is executed in the batinclude section is

```
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\DIMDISP.GMS
  86  display 'eee is 0 dimensional',eee;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\DIMDISP.GMS
 100  display 'set1 is 1 dimensional',set1;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\DIMDISP.GMS
 113  display 'fff is 1 dimensional',fff;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\DIMDISP.GMS
 129  option ggg:0:1:3;display 'ggg is  4 dimensional',ggg;
BATINCLUDE C:\GAMS\GAMSPDF\BIGONE\DIMDISP.GMS
143option hhh:0:1:4;display 'hhh is  5 dimensional',hhh;
```

Note that each time the batinclude file is called that only one of the 11 different display statements becomes active. Also note that the one that is chosen corresponds to the dimension of the named item used as an argument in the batinclude.

14.4.5.3 Passed parameter existence

Conditionals may be structured to examine whether or not one of the allowable parameters passed as an argument into the include family of commands is in fact omitted or present. In such case, one can use text comparison syntax as I discussed above and enter a statement of the following form.

```
$If "a%5" == "a" Display 'argument 5 is blank;
```

The text on the left hand side of the == comparison will equal just "a" if the fifth argument in the batinclude is null and will then equal "a" causing the conditional to be true.

14.4.5.4 Based on GAMS command line parameters

One can use a $If or $Ifi with or without the not modifier to test the characteristics of the GAMS command line parameter appearing on the GAMS call or in the GAMS parameters box in the upper right corner of the IDE. Such items can be addressed as attributes of the keyword GAMS as follows

```
%Gams.commandlineparametername%
```

The parameter names are all listed in the Command Line Parameters chapter.

Example:

(gamsparm.gms)

When I include the statements

```
$If NOT '%gams.lp%'     == '' $set lp     %gams.lp%
$If NOT '%gams.rmip%'   == '' $set rmip   %gams.rmip%
$If NOT '%gams.mip%'    == '' $set mip    %gams.mip%
$If NOT '%gams.nlp%'    == '' $set nlp    %gams.nlp%
$If NOT '%gams.dnlp%'   == '' $set dnlp   %gams.dnlp%
```

© 2022 Prof. Bruce McCarl

```
$If NOT '%gams.cns%'    == '' $set cns    %gams.cns%
$If NOT '%gams.mcp%'    == '' $set mcp    %gams.mcp%
$If NOT '%gams.rminlp%' == '' $set rminlp %gams.rminlp%
$If NOT '%gams.minlp%'  == '' $set minlp  %gams.minlp%
$If NOT '%gams.lp%'   == '' display 'lp     command argument used %gams.lp%';
$If NOT '%gams.rmip%' == '' display 'rmip   command argument used %gams.rmip%';
$If NOT '%gams.mip%'  == '' display 'mip    command argument used %gams.mip%';
$If NOT '%gams.nlp%'  == '' display 'nlp    command argument used %gams.nlp%';
$If NOT '%gams.dnlp%' == '' display 'dnlp   command argument used %gams.dnlp%';
$If NOT '%gams.cns%'  == '' display 'cns    command argument used %gams.cns%';
$If NOT '%gams.mcp%'  == '' display 'mcp    command argument used %gams.mcp%';
$If NOT '%gams.rminlp%'=='' display 'rminlp command argument used %gams.rminlp%
$If NOT '%gams.minlp%'== '' display 'minlp  command argument used %gams.minlp%'
$If NOT '%gams.ps%' == '' display 'Page size command argument used %gams.ps%';
$If NOT '%gams.pw%' == '' display 'Page width command argument used %gams.pw%';
$show
```

and use the GAMS command line parameters

```
lp=bdmlp mip=osl nlp=conopt ps=60 pw=85
```

the resultant code in the compiler after resolving the above statements that is passed on to execution along with the report on the control variables becomes

```
  3  option lp=cplex;
 13  display 'lp     command line argument used bdmlp';
 15  display 'mip    command line argument used osl';
 16  display 'nlp    command line argument used conopt';
 22  display 'Page size  command line argument used 60';
 23  display 'Page width command line argument used 85';

---- Begin of Environment Report
LEVEL TYPE         LINE  FILE NAME
----------------------------------
    0 INPUT          23  C:\GAMS\GAMSPDF\GAMSPARM.GMS

LEVEL SETVAL                          TYPE        NUM TEXT
-------------------------------------------------------------
    0 lp                              SCOPED        1 bdmlp
    0 mip                             SCOPED        1 osl
    0 nlp                             SCOPED        1 conopt
---- End of Environment Report
```

Notes:

- The only non blank items that can be found within the Gams.commandlineparametername entries are those explicitly entered on the command line of the DOS level GAMS command or in the command line parameters box in the IDE.
- Internal option statements addressing the same items have no effect on the

© 2022 Prof. Bruce McCarl

Gams.commandlineparametername entries.

- Default values for the parameters will not be reported.

14.4.5.5 Based on system characteristics

One can use a $If or $Ifi with or without the not modifier to test the characteristics of a number of GAMS system options. Such items are addressed as an attribute of the keyword system as follows

```
%system.attribute%
```

Most of the available words are not useful in conditionals, thus I list the attributes further below and only deal with the Filesys attribute.

That syntax is

```
$If %system.filesys% == type   gamsstatement
```

where type can be DOS, MS95, UNIX, MSNT and others that identify the type of system on which the job is being run. This allows operating specific command inclusion.

Example:

(comparw.gms)

When I include the statements

```
$set console
$If %system.filesys% == UNIX  $set console /dev/tty
$If %system.filesys% == DOS   $set console con
$If %system.filesys% == MS95  $set console con
$If %system.filesys% == MSNT  $set console con
$If "%console%." == "." abort "filesys not recognized";
file screen / '%console%' /;
```

that allow definition of a put file destination that is dependent upon the operating system that is being used. In this case I am setting control variable console to the operating system dependent name of the screen that users can view when a job is running. In turn, this permits us to use put commands to pass information to the user on program execution progress during the conduct of a time consuming procedure.

14.4.5.6 Based on error and warning checks

Conditionals may be placed in the GAMS code that do particular things only if compilation up until that point has been

- free of errors
- a call of an external program resulted in an error or not
- Free of warnings

Such commands are of the form

© 2022 Prof. Bruce McCarl

```
$If errorfree gamsstatement
$If errorlevel n gamsstatement
$If warnings gamsstatement
```

where

- The first syntax alternative is true if the compilation has been error-free up to the point at which the statement appears.
- The second syntax alternative is true if the return code of a program is equal to n. n specifies the return code value. Therefore if 1 was used, the conditional will be true if the return code value was 1.
- The third syntax alternative is true if the compilation has been warning free up to the point at which the statement appears.

14.4.5.7 Based on file or directory existence

Conditionals may be placed in the GAMS code that do particular things only if a named file or directory exists or conversely with the use of the not command if it does not exist. Such commands are of the form

```
$If exist fullfilename gamsstatement
$If not exist fullfilename gamsstatement.
$If dexist fulldirectoryname gamsstatement
$If not dexist fulldirectoryname gamsstatement.
```

where the first of the four syntax alternatives yields a true conditional if the file exists and the second if it does not. Similarly the third of the four syntax alternatives yields a true conditional if the directory exists and the last if it does not

Examples:

```
$If exist c:\myfile $call "copy "c:\myfile con"
$If exist fileexist.gms $call "copy fileexist.gms con"
$If dexist c:\ $call "copy "c:\myfile con" c:\
```

These statements also show one can include operating system commands as discussed below.

14.4.5.8 Based on put file status

Conditionals may be placed in the GAMS code to do particular things only if some put file is open. Such commands are of the form

```
$If putopen gamsstatement
$If not putopen gamsstatement
```

where the first of the two syntax alternatives yields a true conditional if both a file statement and at least one put statement have been executed. Note this does not guarantee that a file will be open at runtime.

© 2022 Prof. Bruce McCarl

Example:

(putopen.gms)

```
$If putopen $goto around
file myfile;
put myfile;
$label around
```

14.4.6 Incorporating Goto: $Goto and $Label

The $If syntax alternatives above only allows conditional execution of a single line of GAMS commands (actually that line can contain several individual commands separated by one or more ;). This can be an obstacle and can be overcome using the $Goto and $Label syntax. Specifically, one can incorporate commands like

```
$Goto labelname
```

within a $If conditional or in it's own line in the GMS file that causes branching to a place where the following command appears

```
$Label labelname
```

which identifies a place to which the code can branch.

Note $IFTHEN and variants can get used if one wishes to jump multiple statements.

Example:

(goto.gms)

```
scalar y /1/;
$Setglobal gg
$If setglobal gg $goto yesgg
y=y+3;
display y;
$label yesgg
display y;
*after yesgg
$If not setglobal gg $goto nogg
y=y/14;
display y;
$label nogg
```

The effect on the code is

```
3  scalar y /1/;
6  *after yesgg
8  y=y/14;
```

© 2022 Prof. Bruce McCarl

```
9  display y;
```

Note the red lines are suppressed because the true $If causes the $Goto to skip around.

14.4.7 Redefining expressions

It is possible to alter the contents of lines to be compiled by including text strings from system attributes, GAMS attributes, passed parameters and control variables. The syntax via which one can include this information is discussed below.

System attributes that can be included
GAMS command line attributes that can be included
Passed parameter inclusion
Control variable inclusion

14.4.7.1 System attributes that can be included

System attributes may be used in conditional compilation. The characteristics of the various attributes are most comprehensively discussed in the Output via Put Commands chapter. The ones that can be used and a brief description follows

.DATE	Identifies date on which model was run
.ELAPSED	Identifies time used
.ERRORLEVEL	Identifies error level
.FE	Identifies file extension of input file
.FN	Identifies file name stem of input file
.FP	Identifies file path of input file
.FILESYS	Identifies name of the operating system being used in
.GAMSRELEASE	Identifies GAMS release number
.GAMSVERSION	Identifies GAMS version number
.GSTRING	Identifies exact GAMS version being used in
.IFILE	Identifies main input file
.INCPARENT	Identifies parent file that includes this one
.INCNAME	Identifies name of file being included
.INCLINE.	Identifies line number of include file being executed
.LICE1	Identifies first license file line
.LICE2	Identifies second license file line
.LINE	Identifies line number of overall file being executed
.LISTLINE	Identifies listing file line number
.LICENSESTATUS	Identifies if a license error has arisen (returns a nonzero in such a case)
.LICENSESTATUSTEXT	Gives text sting that describes a license error)if one arose)
.LP,NLP,...	For all model types identifies solver
.MEMORY	Identifies memory used
.OFILE	Identifies LST file

© 2022 Prof. Bruce McCarl

.OPAGE	Identifies page number in output
.PAGE	Identifies output page
.PFILE	Identifies current put file
.PLATFORM	Identifies computer type
.PRLINE	Identifies line in output file
.PRPAGE	Identifies page in output file
.RDATE	Identifies run date
.REDIRLOG	Identifies log file name
.RFILE	Identifies restart file
.RTIME	Identifies restart file creation time
.SFILE	Identifies save file
.SSTRING	Identifies last solver used
.TCLOSE	Identifies time job ended
.TCOMP	Identifies compile time
.TEXEC	Identifies execution time
.TIME	Identifies time of run
.TITLE	Identifies Job title
.TSTART	Identifies time job started
.VERSION	Identifies GAMS version number
.VERID	Identifies GAMS version

These attributes may be included in the command soon using the syntax %system.attribute % or "%system.attribute%" where the quoted form is preferable if there are spaces in the line.

Examples:

(syschar.gms)

```
$set systemDATE      "%system.DATE%"
$set systemTIME      "%system.TIME%"
$set systemINCPARENT "%system.INCPARENT%"
$set systemINCNAME   "%system.INCNAME%"
$set systemINCLINE   "%system.INCLINE%"
$set systemLINE      "%system.LINE%"
$set systemVERSION   "%system.VERSION%"
$set systemGSTRING   "%system.GSTRING%"
$set systemFILESYS   "%system.FILESYS%"
$set systemPRLINE    "%system.PRLINE%"
$set systemPRPAGE    "%system.PRPAGE%"
$show
display "system.DATE"      ,"%system.DATE%"      ;
display "system.TIME"      ,"%system.TIME%"      ;
display "system.INCPARENT" ,"%system.INCPARENT%"     ;
display "system.INCNAME"   ,"%system.INCNAME%"      ;
display "system.INCLINE"   ,"%system.INCLINE%"      ;
display "system.LINE"      ,"%system.LINE%"      ;
display "system.VERSION"   ,"%system.VERSION%"      ;
display "system.GSTRING"   ,"%system.GSTRING%"      ;
display "system.FILESYS"   ,"%system.FILESYS%"      ;
display "system.PRLINE"    ,"%system.PRLINE%"      ;
display "system.PRPAGE"    ,"%system.PRPAGE%"      ;
```

© 2022 Prof. Bruce McCarl

14.4.7.2 GAMS command line attributes that can be included

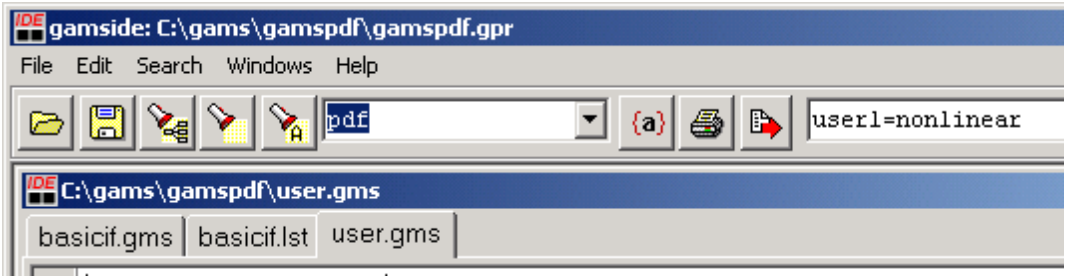
Parameters used in the GAMS command line may be used in conditional compilation. The usage of the various attributes involves the syntax %gams.attribute % or "%gams.attribute%" where the quoted form is preferable if their spaces in the attribute. This is illustrated above in the section on conditionals based on GAMS command line parameters. The variety of options available is given in the Command line parameters chapter.

14.4.7.2.1 Based on user options and command line: -- // -/ /- User1-5

(user.gms)

One can define global control variables and user input variable through the command line and then use them in controlling the gams code as discussed in the conditional compilation chapter. In particular one can use -- , //, -/ , or /- parameters plus user1 through user5 in the GAMS call or the parameter box of the IDE.

For example if one puts --mymodel=yesnonlinear user1=nonlinear in the control box I can treat a model as either linear or nonlinear depending on the mymodel global variable or the user1 setting



and when the user1 setting is coupled with the following (nlplp.gms)

```
$If not "%mymodel%" == "yesnonlinear" $goto arond
$If not "%gams.user1%" == "nonlinear" $goto arond
```

then the NLP form is controlled from command line

```
gams myrun --mymodel=yesnonlinear user1=nonlinear
```

One can also use user2, user3, user4, user5. The abbreviation for these items is U1, U2, U3, U4, U5

More generally the setglobal variable syntax is any of the following four

```
--name=string
//name=string
/-name=string
-/name=string
```

where name is the name of a control variable chosen by the user and string a text string.

© 2022 Prof. Bruce McCarl

For example (**minusminus.gms**) one could use

```
--keycity=Boston //myflag=modes /-myvalue=7.6 -/dothis="display x;"
```

specifying the control variables keycity, myflag, myvalue and dothis with the strings Boston, modes, 7.6 and display x; respectively. In tune when one had a statement

```
x("%keycity%","%myflag%")=%myvalue%;
%dothis%
```

it would become

```
x("boston","mode")=7.6;
display x;
```

GAMS checks for proper spelling and definition of these items when the dollar command $setddlist is entered in the code.

14.4.7.3 Passed parameter inclusion

Parameters passed into batinclude files may be used in conditional compilation as discussed in the in the Including External Files chapter under the parameter substitution section. The usage of the various attributes involves the syntax %n or "%n" where n is the number of the parameter in the argument list. The quoted form is preferable if text with spaces is to be included. This is illustrated above in the section on conditionals based on characteristics of named item or parameter.

14.4.7.4 Control variable inclusion

The text in control variables may be incorporated into statements via conditional compilation. The usage of the various attributes involves the syntax %controlvariablename% or "%controlvariablename%". The quoted form is preferable if the control variable contains text with spaces. This is illustrated above in the section on conditionals based on control variable contents and in the example **gamsparm.gms**.

14.4.8 Running external programs or commands

One can use the $If tests in conjunction with the $Call and Execute commands to optionally include execution of external files.

$Call
Execute
Shellexecute
$Setargs

14.4.8.1 $Call

This command uses the syntax

© 2022 Prof. Bruce McCarl

```
$Call commandtoexecuteinOS
```

or

```
$Call.Async commandtoexecuteinOS
```

to execute a program or operating system command specified by commandtoexecuteinOS during compilation.

- When used without the suffix it halts compilation until that file has run successfully. Thus one can create a file and then include in into the program.
- When used with the .Async suffix it continues without waiting for the file to run successfully.

Use of $Call is discussed in the Links to Other Programs Including Spreadsheets chapter.

14.4.8.2 Execute

This command uses the syntax

```
Execute commandtoexecuteinOS
```

To executes a program or OS command specified by commandtoexecuteinOS during GAMS execution. Use of $Call is discussed in the Links to Other Programs Including Spreadsheets chapter.

14.4.8.3 Shellexecute

Allows execution of a program chosen by the operating system given a file name. This is discussed here.

14.4.8.4 $Setargs

One can use $setargs args to set arguments for batinclude calls as discussed in the Including External Files chapter.

14.4.9 Writing messages to LST, LOG and other files

Users may also write conditional messages to the Lst, Log and other files.

LST File: $Abort and $Error
LOG file: $Log
Other named files: $Echo, $Offecho, $Onecho

14.4.9.1 LST File: $Abort and $Error

Messages to the LST file may be written on program termination. To stop compilation and write a message use

```
$abort message
```

© 2022 Prof. Bruce McCarl

while

```
$error message
```

generates a compiler error and outputs the attached message.

$Echo can also be used as discussed below. There is also an execution time Abort that also writes messages as discussed in the Conditionals chapter.

14.4.9.2 LOG file: $Log

Messages to the LOG file may be written at any time. Use of the syntax

```
$log message
```

sends the text message to the log file.

14.4.9.3 Other named files: $Echo, $Offecho, $Onecho

Messages to any named file can be written using $Echo, $Onecho and $Offecho. The $Echo sends one line and is invoked using the syntax

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'text to be sent' >> externalfile
```

where external file can be of unlimited length.

For multi line messages

```
$onecho  > externalfile
line 1 of text to be sent
line 2 of text to be sent
...
last line of text to be sent
$offecho
```

Notes:

- Both the text and the file name can be quoted or unquoted.
- The file name by default will go in the working directory.
- The file is not closed until the end of the compilation or when a $call or any kind of $include statement is encountered.
- The redirection symbols > causes any files with the same name to be overwritten.
- The redirection symbols >> causes any files with the same name to be appended to.

© 2022 Prof. Bruce McCarl

- An example is at bottom of **condcomp.gms**.
- One can also use echo to display environment variables as discussed above.

14.4.10 End the job: $Exit, $Abort, $Error, $Stop, $Terminate

One can cause the job to terminate in several ways

To exit compilation use

```
$exit
```

To stop compilation and write a message use

```
$abort message
```

To generate a compile error with a message use

```
$error message
```

To stop without a message use

```
$stop
```

To stop compilation and end the job

```
$terminate
```

14.4.11 Longer examples

Here I present some examples illustrating various usages.

Changing model type depending on control variable
Changing form of data in model and their use
Having batincludes that deal with different data types
For more examples

14.4.11.1 Changing model type depending on control variable

Here I set up a model as either a linear or nonlinear form depending on the control variable nonlin (**nlplp.gms**).

```
$Setglobal nonlin yes
*$Setglobal nonlin no
variables           z   objective
```

© 2022 Prof. Bruce McCarl

```
positive variables   x   decision variables;
equations            obj
                     xlim;
$If %nonlin% == yes $goto nonlin
        obj..   z=e=3*x;
$goto around
$label nonlin
        obj..   z=e=3*x-3*x**2;
$label around
        xlim..   x=l=4;
model cond /all/;
$If %nonlin% == yes solve cond using nlp maximizing z;
$If not %nonlin% == yes solve cond using lp maximizing z
```

When NONLIN is set to yes I get

```
 3  variables            z   objective
 4  positive variables   x   decision variables;
 5  equations            obj
 6                       xlim;
 8          obj..   z=e=3*x-3*x**2;
10          xlim..   x=l=4;
11  model cond /all/;
13  solve cond using nlp maximizing z;
```

otherwise

```
 3  variables            z   objective
 4  positive variables   x   decision variables;
 5  equations            obj
 6                       xlim;
 8          obj..   z=e=3*x;
10          xlim..   x=l=4;
11  model cond /all/;
13  solve cond using lp maximizing z;
```

14.4.11.2 Changing form of data in model and their use

Here I have a transport model that either works with or without consideration of transport modes depending on the control variable called mode as illustrated in **mode.gms**.

```
$Setglobal mode
Sets Source       plants   / Seattle, "San Diego" /
     Destinaton   markets          / "New York", Chicago, Topeka / ;
Parameters  Supply(Source)   Supply at each source
           /seattle  350, "san diego" 600  /
            Need(Destinaton) Demand at each market
           /"new york" 325, chicago   300,  topeka 275  / ;
```

© 2022 Prof. Bruce McCarl

```
Table distance(Source,Destinaton)  distance in thousands of miles
                    "new york"      chicago      topeka
      seattle          2.5           1.7          1.8
      "San diego"      2.5           1.8          1.4  ;
$If setglobal mode $goto mode
Scalar  prmilecst   freight cost in $ per case per 1000 miles /90/
        loadcost    freight loading cost in $ per case /25/        ;
Parameter trancost(Source,Destinaton)  transport cost in dollars per case ;
   trancost(Source,Destinaton) =
         loadcost + prmilecst * distance(Source,Destinaton) ;
$goto around
$label mode
set mode /truck,train/
parameter prmilecst(mode)  /truck 90,train 70/
          loadcost(mode)   /truck 25,train 100/        ;
Parameter trancost(Source,Destinaton,mode)  transport cost  ;
   trancost(Source,Destinaton,mode) =
         loadcost(mode) + prmilecst(mode) * distance(Source,Destinaton) ;
$label around
Positive Variable
$If setglobal mode transport(Source,Destinaton,mode) shipment quantities in case
$If not setglobal mode transport(Source,Destinaton) shipment quantities in cases
Variable    totalcost total transportation costs in dollars ;
Equations   Costsum                 total transport cost -- objective function
    Supplybal(Source)       supply limit at source plants
    Demandbal(Destinaton)   demand at destinations ;
$If not setglobal mode $goto nomode
Costsum ..  totalcost  =e=  sum((Source,Destinaton),
          sum(mode,trancost(Source,Destinaton,mode)
                  *transport(Source,Destinaton,mode)));
Supplybal(Source) ..
     sum((destinaton,mode), transport(Source,Destinaton,mode))
            =l=  supply(Source) ;
demandbal(Destinaton) ..
   sum((Source,mode), transport(Source,Destinaton,mode))
           =g=  need(Destinaton) ;
$goto modset
$label nomode
Costsum ..  totalcost  =e=  sum((Source,Destinaton),
                 trancost(Source,Destinaton)
                 *transport(Source,Destinaton));
Supplybal(Source) ..
     sum((destinaton), transport(Source,Destinaton))
            =l=  supply(Source) ;
demandbal(Destinaton) ..
   sum((Source), transport(Source,Destinaton))
           =g=  need(Destinaton) ;
$label modset
  Model tranport /all/ ;
  Solve tranport using lp minimizing totalcost ;
```

© 2022 Prof. Bruce McCarl

14.4.11.3 Having batincludes that deal with different data types

Here I write some code that puts data using put commands that can either be a set or a parameter and figures out which then uses the put commands (**putcond.gms**).

```
file at
put at
set a1 set to be put/item1 first,item2 second/
parameter r(a1) parameter to be put  /item1 5,item2 6/
$batinclude outit a1
$batinclude outit r
```

where **outit.gms** is

```
$If not "a%1" == "a" $goto start
$error Error in outit: item to be printed is not specified.
$label start
$If declared %1    $goto declared
$error Error in outit: identfier %1 is undeclared.
$exit
$label declared
$If defined %1     $goto defined
$error Error in outit: identfier %1 is undefined.
$exit
$label defined
$If settype %1  $goto doset
$If partype %1  $goto dopar
$error Error in outit: identfier %1 is not a set or a parameter.
$exit
$label doset
put /' set %1  ' %1.ts /
loop(%1,put '    Element called ' %1.tl ' defined as ' %1.te(%1) /)
put  /
$goto end
$label dopar
$If not dimension 1 %1 $goto badnews
$If not declared wkset1 alias(wkset1,*);
$If not declared wkset2 set wkset2(wkset1);
wkset2(wkset1)=no;
$onuni
wkset2(wkset1)$%1(wkset1)=yes;
display wkset2;
put /' Parameter %1  ' %1.ts  /
loop(wkset2,put '    Element ' wkset2.tl ' equals ' %1(wkset2) /)
put  /
$offuni
$goto end
$label badnews
$error Error in outit: identfier %1 is not a one dimensional parameter.
```

© 2022 Prof. Bruce McCarl

```
$label end
```

which becomes

```
   1  file at
   2  put at
   3  set a1 set to be put/item1 first,item2 second/
   4  parameter r(a1) parameter to be put  /item1 5,item2 6/
BATINCLUDE C:\GAMS\ADVCLASS\CLASS\EXAMPLE\CONDCOMP\OUTIT.GMS
  10  put /' set a1  ' a1.ts /
  11  loop(a1,put 'Element called' a1.tl 'definedas' a1.te(a1) /)
  12  put  /
BATINCLUDE C:\GAMS\ADVCLASS\CLASS\EXAMPLE\CONDCOMP\OUTIT.GMS
  21  alias(wkset1,*);
  22  set wkset2(wkset1);
  23  wkset2(wkset1)=no;
  25  wkset2(wkset1)$r(wkset1)=yes;
  26  display wkset2;
  27  put /' Parameter r  ' r.ts  /
  28  loop(wkset2,put ' Element' wkset2.tl 'equals ' r(wkset2) /)
  29  put  /
```

Note lines 21-25 figure out the set elements r is defined over and put it in set wkset2.

14.4.11.4 For more examples

For more examples see **gnuplotxyz.gms**, or Rutherford's http://www.mpsge.org/gnuplot.

15 Using GAMS Data Exchange or GDX Files

GAMS can read or write something called a GDX file. The name GDX is an acronym for GAMS Data eXchange files. A GDX file is a platform independent, binary file that can contain information regarding sets, parameters, variables and equations. Among other usages GDX files can be used to prepare data for a GAMS model, pass results of a GAMS model into different programs, and pass results into GAMS from different programs. Additional information may be found in the GAMS document *GDX in GAMS* at GAMS.

Creating a GDX file in GAMS
Inputting data from a GDX file into GAMS
General notes on GDX files
Identifying contents of a GDX file
Using GDX files to interface with other programs
Compressed GDX files

© 2022 Prof. Bruce McCarl

15.1 Creating a GDX file in GAMS

A GDX file can be created by GAMS in three alternative forms

A total problem summary GDX file may be created
A GDX point file of solution information can be created
A selected item GDX file may be created

Such files are only created on explicit user request although this may be indirect when a program like Xlexport is included which in turn creates a GDX file.

Now let's review these cases.

Command line GDX option - GDX dump of the whole problem
GDX Point Solution file
GDX files containing selected items

15.1.1 Command line GDX option - GDX dump of the whole problem

A composite GDX file containing all data items resident at the end of the run of a GAMS code can be created using the command line GDX parameter. The command line GDX option is invoked by adding the option GDX=filename to the GAMS command line in DOS or Unix/Linux or by including it in the command line parameter box in the IDE. The basic command line form is

```
gams mymodelname GDX=gdxfilename
```

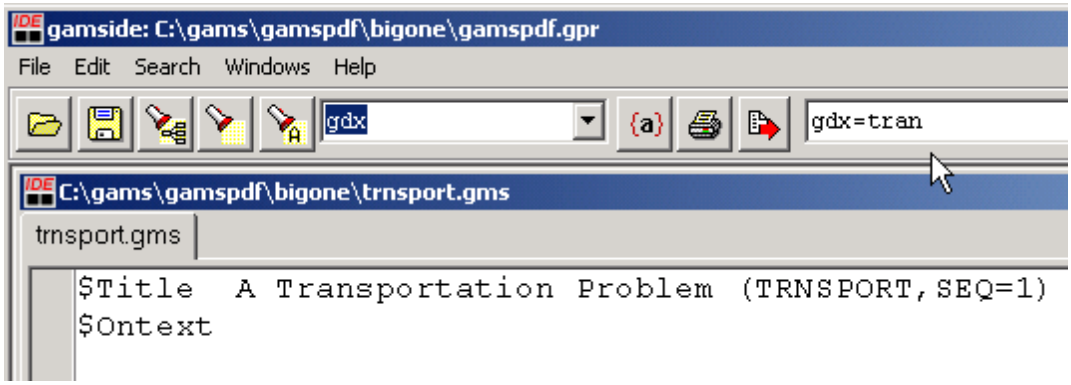
where

- mymodelname specifies the name of the file of GAMS instructions
- gdxfilename gives the file name and possible path where the GDX file is to be retained. When no path is specified the default directory is the current working directory or project directory in the IDE as below.
- Setting GDX to the sting "default" ie gdx=default causes GAMS to create aGDX file with the GMS file root name and a GDX extension. Thus gams trnsport gdx=default will cause GAMS to write the reference file trnsport.gdx.

Example:

An example of DOS invocation of the whole problem GDX file for the **trnsport.gms** model is given in **gamsgdx.bat**. When the IDE is used, the GDX file creation is invoked by an entry in the upper right hand corner of the IDE screen as illustrated below

© 2022 Prof. Bruce McCarl



Notes:

- When this option is used the GDX file is created just at the end of the GAMS execution so the data written will contain the current values for all sets, parameters, variables and equations that are on hand at the end of the GAMS job.
- The GDX data for the variables and equations contains the levels, marginals, lower bounds, upper bounds and scales for each item.
- This yields a file that may be automatically opened in the IDE by doing a mouse click on the highlighted line in the IDE process window.

15.1.2 GDX Point Solution file

A GDX file containing the marginals and levels for all variables and equations at the end of a solve will be created with the command line parameter, model attribute or option Savepoint before the solve is invoked. One can save the solution information from the last solve or from every solve. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point.

The basic command line form is

```
gams mymodelname Savepoint=number
```

the model attribute form is

```
modelname.savepoint=number;
```

and the option file form is

```
option savepoint=number
```

where

- when number equals 1 a point gdx file is saved from the last solve in the GAMS model and the file name will be modelname_p.gdx where model name is the name of the model identified in

© 2022 Prof. Bruce McCarl

the solve statement.

- when number equals 2 (**gdxsavepoint2.gms**) a point gdx file is saved from the every solve in the GAMS model and the file name will be modelname_pnn.gdx where model name is the name of the model identified in the solve statement and nn is the internal number of the solve. Thus if 10 solves occur one will get 10 files named modelname_p1.gdx through modelname_p10.gdx.

Example:

An example of invocation of the GDX point file is given in **makepointbas.gms**, the relevant part of which is

```
MODEL FIRM /ALL/;
OPTION Savepoint=1;;
solve firm using LP maximizing objfun;
```

and the file **gdxsavepoint2.gms** illustrates the case where savepoint is set to 2.

Notes:

- the GDX point file contains numerical records for all variables and equations giving just their levels and marginals. In a non point GDX file information on bounds and scales are also present, if defined.
- This GDX point file can be reloaded into GAMS using either the Execute_loadpoint or Execute_load or $load syntaxes.
- This yields a file that may be automatically opened in the IDE by doing a mouse click on the highlighted line in the IDE process window.

15.1.3 GDX files containing selected items

Selected items may be placed into a GDX file either at compile time or during execution. The syntax and effects differ so these are discussed separately.

15.1.3.1 Execution time selected item GDX file creation

An Execute_Unload or a Execute_Unloaddi command creates a GDX file containing selected problem data. The data in the GDX file are those present at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected.

The basic syntax of the statements are

```
Execute_Unload   'filename', nameditem1,nameditem2, ... ;
Execute_Unloaddi 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the resultant GDX file. In particular, a file with this name is created with the extension .GDX and is placed in the current working directory. This opens and closes the GDX file and does all the writing. Note the Execute_Unload command overwrites any existing file with the name filename.gdx so all writing to the file must be done in one statement.

© 2022 Prof. Bruce McCarl

The second part of the statement is a list of items to be placed in the GDX file and has several variants. For example, one could use multiple lines and unload several items with the command structure

```
Execute_Unload 'filename',   nameditem1
                                    nameditem2,
                                    nameditem3
                                    nameditem4 ;
```

When Execute_Unload is used then just the named items are in the GDX file. When the alternative **Execute_Unloaddi** is used then all the sets that the named items are defined over are also included in the GDX file

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case, the syntax is

```
Execute_Unload 'filename', internalname1=GDXitemname1 i2=gf2;
```

and would result in the GAMS item called internalname1 being called gdxitemname1 in the GDX file and i2 being called gf2. This syntax again can be repeated for multiple items.

Finally when the Execute_Unload is run without any items named then the GDX file automatically contains all items in the GAMS program, ie

```
Execute_Unload 'filename'
```

will cause filename to contain all sets, parameters etc.

Example:

In the model **gdxexectrnsport.gms** we introduce the statement

```
execute_unload 'tran2',i,j,d,f,a=sup,b=dem,x,supply;
```

The result of this is the writing of the GDX file **tran2.gdx** that contains the data for the sets i and j plus the parameters d, f, a and b as well as the variables x and the equations supply. In that file the a and b items have been renamed and are identified as sup and dem.

We also add the statement

```
execute_unloaddi 'tran2di',d, x, item;
```

which does not have any set names specified within it just parameter and variable names. However the resultant GDX file **tran2di.gdx** contains the data for the parameters d, and itemplus variable x. However it also contains the sets i, j and k since the other items are defined over them.

Notes

- The name of the active file being unloaded to can be changed with the Put_utility 'gdxout' syntax

© 2022 Prof. Bruce McCarl

15.1.3.2 Compile time selected item GDX file creation

A group of dollar commands can be used to write a GDX file containing selected data. The data written to the GDX file will be those present when the statement is encountered during compilation. The results of calculations and solves will not be reflected. *(Note this should not ordinarily be used, it is safer to use the Execute_Unload as calculations and solves would be reflected in the result).* The only way to guarantee that the data is current is to use the execution time command or to use a save then restart a file with the dump commands within them.

The basic syntax involves a three-part sequence

```
$Gdxout filename
$Unload itemname
$Gdxout
```

The first part of the sequence is the initial $Gdxout command which also specifies the filename that the GDX file will be called. A file with this name will be placed in the current working directory with the extension .GDX. This opens the GDX file and prepares it for writing. Any existing files with the same name will be overwritten.

The second part of the sequence is one or more $Unload commands. These commands specify the items to be placed in the GDX file. A statement can specify more than one item. For example, one could unload four items with the following commands

```
$Gdxout filename
$unload itemname1
$unload itemname2
$unload itemname3
$unload itemname4
$Gdxout
```

or could accomplish the same using

```
$Gdxout filename
$unload itemname1 itemname2 itemname3 itemname4
$Gdxout
```

It is also possible to have different names for parameters in the GDX file as opposed to the names used in the GAMS program. In such a case the syntax is

```
$unload internalname1=gdxfileitemname1 i2=gf2
```

which would result in the item with internalname1 being called gdxfileitemname1 in the GDX file and i2 being called gf2.

The third part of the sequence simply consists of a $Gdxout command which closes the GDX file. Actually the statements can be intermixed with GAMS calculations solves etc. but must eventually be closed with a $Gdxout.

© 2022 Prof. Bruce McCarl

Example:

In the model **gdxtrnsport.gms** we introduce the sequence

```
d(i,j)=d(i,j)*10;
$GDXout tran
$unload i j
$unload d
$unload f
$unload a=dem b=sup
$GDXout
```

The result of this is a GDX file named **tran.gdx** that contains the data for the sets i and j as well as the parameters d, f, a and b. Note that the a and b items have been renamed dem and sup. Also note the d items will not have been multiplied by 10 but rather take on their compile time values.

15.2 Inputting data from a GDX file into GAMS

Data in a GDX file can be read during a GAMS compile or a compile/execute sequence. GAMS can only load data from GDX files into declared items and only on an item-by-item basis. In addition GDX files are read when Ximport is included which in turn runs a program that creates a GDX file with Excel contents and then Ximport reads the Excel data in that GDX file.

Items may be loaded at compile time or during execution. The syntax differs depending on whether items are read at compile or execution time so these are discussed separately.

Compile time imports from GDX files
Execution time GDX imports

15.2.1 Compile time imports from GDX files

A set of dollar commands can be used to cause GAMS to read data from a GDX file at compile time. The data read from the GDX file will be the data present in it at the time that the compile job is begun.

The basic syntax involves a three-part sequence

```
$Gdxin filename
$Load itemname
$Gdxin
```

The first part is an initial $Gdxin command which also specifies the filename to be used. A file with this filename and the extension .GDX is looked for in the current working directory. In turn this command opens the GDX file and prepares it for reading.

The second part of the sequence is one or more $Load commands or the alternative Loaddc (which eploys domain checking). These commands specify the items to be read from the GDX file. Several commands may be used and each line can read more than one item. For example, one could load

© 2022 Prof. Bruce McCarl

several items with the command structure

```
$Gdxin filename
$load itemname1
$loaddc itemname2
$load itemname3
$load itemname4
$Gdxin
```

or could use the structure

```
$Gdxin filename
$load itemname1 itemname2 itemname3  itemname4
$loaddc itemname5 itemname6 itemname7  itemname8
$Gdxin
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case the syntax is

```
$load internalname1=gdxfileitemname1 i2=gf2
```

Any parameter data can be loaded as can set data defining domains and variable/equation data.

The third part of the sequence simply consists of another $Gdxin command which closes the GDX file. Actually the statements can be intermixed with GAMS calculations solves etc. but must eventually be closed with a $Gdxin.

Example:

In the model **gdxintrnsport.gms** we introduce the sequence

```
$gdxin tran2
$LOAD
  Sets
       uni universal set
       i   canning plants
       j   markets;
$load uni=* i j
  Parameters
       a(i)  capacity of plant i in cases
       b(j)  demand at market j in cases;
$load a=sup
$loaddc b=dem
  Parameter d(i,j)  distance in thousands of miles;
$load d
  Scalar f  freight in dollars per case per thousand miles   ;
$load f
$gdxin
```

© 2022 Prof. Bruce McCarl

```
display uni,i,j,a,b,d,f;
```

This loads data from the GDX file named **tran2.gdx** that was saved by the example **gdxexectrnsport.gms**.

Notes:

- Items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the Load appears.
- When using $load GAMS does not domain check ignoring items that are resident in GDX files for named set dependent parameters, variables, equations and sets that do not match current set elements. GAMS will ignore these items and will not create errors or cause generation of any messages.
- When using $loaddc GAMS does domain checking generating errors for items that are resident in the GDX files that do not match internal sets.
- One can import items for set positions that are not in existing sets where the set specified for that position is equivalent to the universal set (i.e. when an * is used or a terms equivalenced to the universal set or the set is a subset of the universal set).
- When the $Load is not followed by arguments this causes a listing of the GDX file contents to be generated.
- One can load the level values of a variable into a parameter of the same dimension using the syntax parametername=var.l as follows

```
parameter storexlevel(i,j);
$gdxin tran2
$LOAD storexlevel=x.l
$gdxin
display storexlevel;
```

- Loading the level values for a variable into the same variable (x=x.l) works but loads everything including bounds, scales, marginals and levels.

15.2.2 Execution time GDX imports

An Execute_Load or an Execute_Loadpoint command can be used to read data from a GDX file. The data in the GDX file will be the data present in the GDX file at the time that the statement is executed and could have been updated by Execute_Unload or Execute_Unloaddi statements or solves under the Savepoint option during the model execution.

15.2.2.1 Execute_Load

When parameter data are loaded using the Execute_Load GAMS acts as if an assignment statement was present, except that it does not merge the data read with the current data; it is a full replacement. Sets defining domains cannot be loaded. However sets that are subsets of existing sets and do not define new elements can be loaded at execution time (Domain defining sets can be loaded can at compile time using $Load).

The basic syntax of the statement is

© 2022 Prof. Bruce McCarl

```
Execute_Load 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the GDX file to read. In particular, a file with this filename with the extension .GDX will be read from the current working directory.

The second part of the statement is a list of items to be read from the GDX file. For example one could load several items with the command structure

```
Execute_Load 'filename',     nameditem1
                             nameditem2,
                             itemname3
                             itemname4 ;
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case the syntax is

```
Execute_Load 'filename',internalname1=GDXitemname1 internalname2=GDXitemname2;
```

Example:

In the model **gdxexecintrnsport.gms** we introduce the statement

```
execute_load 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

The result of this is that the k subset and the parameters are loaded. We also get advanced basis information when we load variables and equations.

Notes:

- Items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the Execute_Load appears.
- When loading data domain checking is not enforced so that when an item is resident in a GDX file for set elements not present in the current file these items are ignored and do not create errors or cause generation of any messages. the Execute_loaddc variant checks to see that the domains match.
- Execute_Load and Execute_Loaddc replaces data in the subject arrays/.
- The name of the active file being loaded from can be changed with the Put utility 'gdxin' syntax
- One can load the universe of labels from a GDX file into a set at run-time using the syntax

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

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

15.2.2.2 Execute_loaddc

Parameter data contained in a GDX file can be loaded with checks to make sure the domain matches up. This is an alternative to using Execute_Load In this case GAMS generates execution errors if the

© 2022 Prof. Bruce McCarl

domain does not match up..

The basic syntax of the statement is

```
Execute_Loaddc 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the GDX file to read. In particular, a file with this filename with the extension .GDX will be read from the current working directory.

The second part of the statement is a list of items to be read from the GDX file. For example one could load several items with the command structure

```
Execute_Loaddc 'filename',   nameditem1
                                        nameditem2,
                                        itemname3
                                        itemname4 ;
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case the syntax is

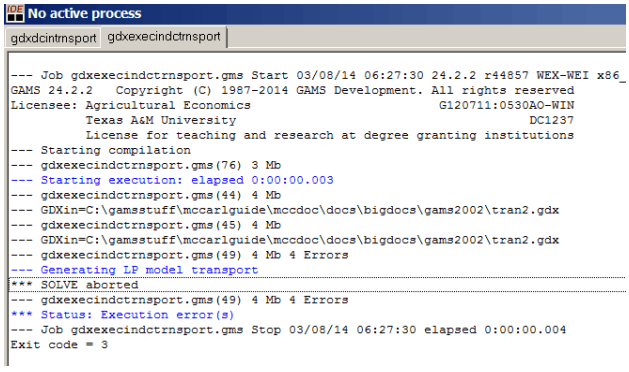
```
Execute_Loaddc 'filename',internalname1=GDXitemname1 internalname2=GDXitemname
```

Example:

In the model **gdxexecindctrnsport.gms** we introduce the statement

```
execute_loaddc 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

but as compared to the file **gdxexecintrnsport.gms** we have deleted the element Topeka from the set definition while it remains in the GDX file. As a consequence we get execution errors



with messages in the list file like

```
**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 1 Domain errors for symbol k
     topeka

**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 2 Domain errors for symbol d
```

© 2022 Prof. Bruce McCarl

```
seattle.topeka
san-diego.topeka
```

The job is aborted..

Notes:

- See the execute_load page for a discussion of basic features.
- Domain errors occur whenever set element names are not spelled exactly the same as an element specified in the corresponding set in GAMS flagging alternative spellings or missing elements.
- Domain errors do not arise when items are not specified with them set to zero (no entery for a set element leaves to a corresponding value of zero
- The job will abort with an execution error

15.2.2.3 Execute_Loadpoint

When Execute_Loadpoint is invoked GAMS goes through the target GDX file looking for variables and equations. The items found are merged into the internal data on those variables and equations replacing the levels and marginals. But

- when variable/equations are present in the GDX file that are not in the current GAMS program that information is ignored.
- when cases are found with set elements that do not match the definitions inside the current GAMS information those data are ignored.
- Bounds, and scales are unaffected.
- Variables in the equations that are GAMS program but not in the GDX file are unaffected by the GDX command.

The basic syntax of the statement is

```
Execute_Loadpoint 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the GDX file to read. In particular, a file with this filename with the extension .GDX will be read from the current working directory. Any GDX file can be read not only a point file (as saved by the savepoint option).

The second part of the statement is an **optional** list of items to be read from the GDX file.

- When a list of items to be read is not present as in the statement (loadpointbas.gms) just below than all variables and equations in the GDX file firm_p will be loaded.

```
execute_loadpoint 'firm_p';
```

- One can also specify the loading of parameters and variables in this context using syntax just like in the Execute_Load syntax above. For example the following command would load the levels for x , the x information into the variable y and the x marginals into the parameter m.

© 2022 Prof. Bruce McCarl

```
execute_loadpoint 'transport_p2' x.l, y=x, m=x.m;
```

Notes:

- Items must be declared before the Execute_Loadpoint appears.
- When loading data domain checking is not enforced so that when an item is resident in a GDX file for set elements not present in the current file these items are ignored and do not create errors or cause generation of any messages.
- The Execute_loadpoint will work on either point or non point GDX files.

- Loadpoint merges data into arrays that are defined
- The name of the active file being loaded from can be changed with the Put utility 'gdxin' syntax

15.3 General notes on GDX files

There are several things worth noting about GDX files

- Only one GDX file can be open at a time.
- When the GDX file to be written has the same name as an existing GDX file the existing file will be overwritten. The resultant file will only contain the new data; there is no merge or append option.
- A compile time GDX write using the $Unload will only write out data defined in the compilation at the point where the command appears. No results of any solves or calculations done within the current GAMS program will be reported with $UnLoad. This is not true with Execute_Unload or Execute_Unloaddi.
- An execution time GDX write using the Execute_Unload or Execute_Unloaddi will write out data defined in the execution sequence at the point where the GDX command appears. The results of the most recent solve command and any parameter calculations occurring before the GDX write will be reported.
- Any subsequent Execute_Unload or Execute_Unloaddi to a file written earlier will totally overwrite that file so care must be taken to write all wanted information in the last appearing Execute_Unload or Execute_Unloaddi.
- A command line GDX write using the GDX=filename command line parameter will write out data defined at the end of the execution sequence. The results of the most recent solve and any parameter calculations will be reported.
- When loading data note that domain checking will not be enforced so that when items are resident in the GDX file for set elements not present in the current file these items will be ignored. GAMS will not generate any message to tell you items are ignored.
- Additional examples of GDX loads and unloads can be found in the library file *gp1x* and in the all the Performance World examples in the *linlib* make use of the Gdxin feature.
- Option Savepoint and Execute_Loadpoint provide a GDX way of saving and loading a basis.
- The contents as they differ between GDX files can be examined with Gdxmerge. or Gdxdiff
- GDX files as of version 22.3 are written in compressed form unless the environment variable

© 2022 Prof. Bruce McCarl

GDXCOMPRESS is set to zero.

- Compressed GDX files are not readable by older GAMS versions but the utility gdxcopy allows one to transform to older versions.
- One can load the universe of labels from a GDX file into a set at run-time using the syntax

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

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

15.4 Identifying contents of a GDX file

Users may wish to examine the contents of a GDX file. However such files are binary and thus do not reveal information if text edited. But the GAMS system provides four ways of accomplishing this, each of which is discussed below.

Identifying contents with $Load
Identifying contents with the IDE
Identifying contents with Gdxdump
Identifying differences in contents with Gdxdiff

15.4.1 Identifying contents with $Load

One can have GAMS tell you the general contents of a GDX file by using the $Load command without the name of a parameter. Namely inserting a sequence like

```
$GDXin tran2
$load
$GDXin
```

yields (**gdxcontents.gms**)

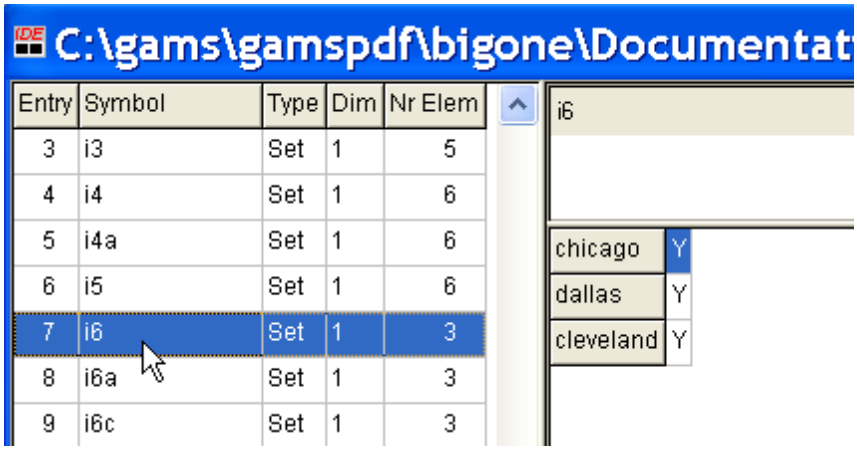
```
Content of GDX C:\GAMS\GAMSPDF\BIGONE\TRAN2.GDX
    Number Type      Dim   Count  Name

      1   Set         1      2    i       canning plants
      2   Set         1      3    j       markets
      3   Parameter   2      6    d       distance in thousands of miles
      4   Parameter   0      1    f       freight in dollars per case per thousa
      5   Parameter   1      2    dem     capacity of plant i in cases
      6   Parameter   1      3    sup     demand at market j in cases
      7   Variable    2      6    x       shipment quantities in cases
      8   Equation    1      2    supply  observe supply limit at plant i
```

which lists the items present by Type, Name, Number of sets the item is defined over(Dim), number of elements in the file for this item (Count).

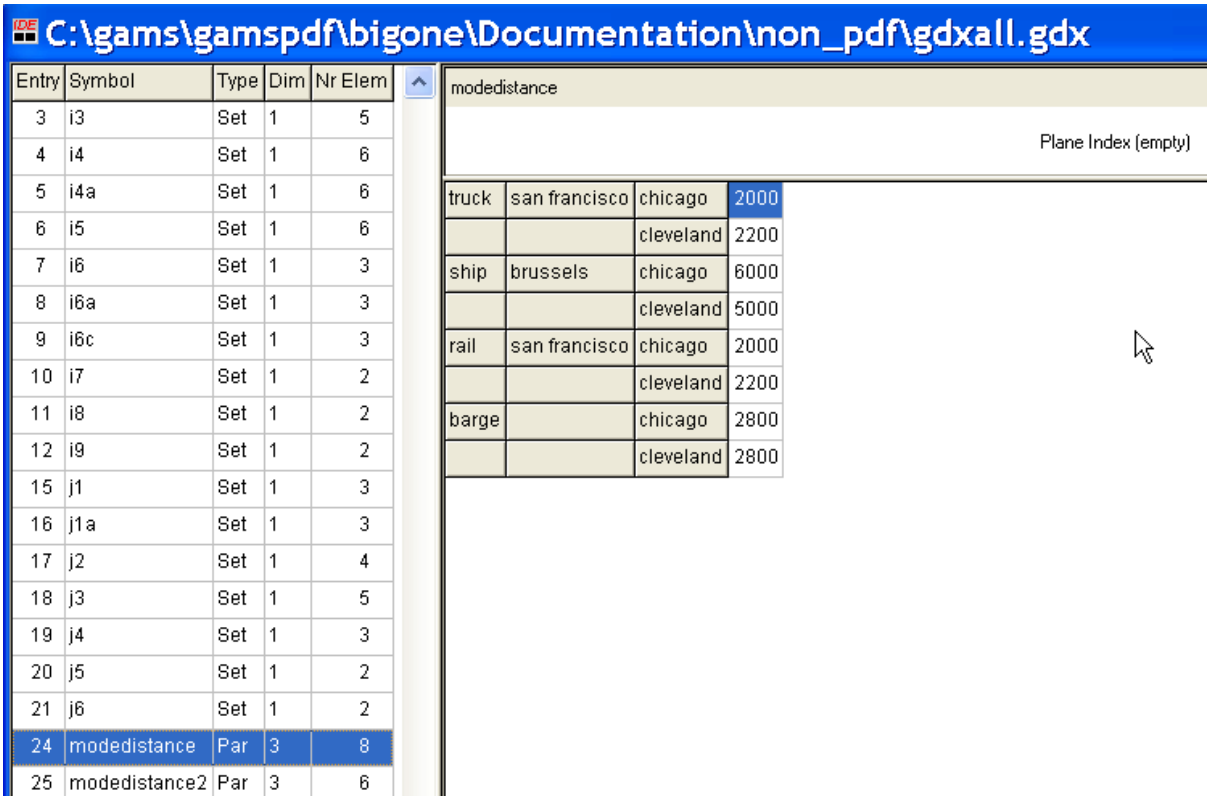
© 2022 Prof. Bruce McCarl

15.4.2 Identifying contents with the IDE

One can use the IDE to view the exact contents of each item in a GDX file by opening a GDX file with the Open file dialogue. The resultant display gives the names of the items in the GDX file on left hand part of the screen while the right part gives the exact data entries for the item highlighted in the left hand part. Namely opening the file **gdxall.gdx** and moving the cursor to the set i6 causes the screens to become

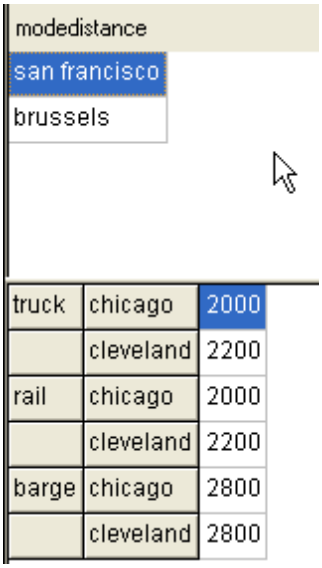


showing the elements in the set as contained within the GDX file. Similarly choosing the object modedistance yields the screen

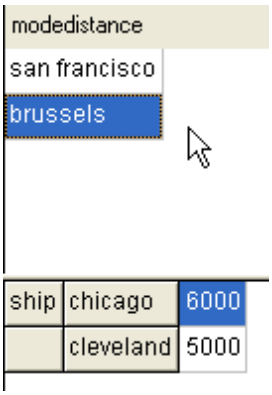


© 2022 Prof. Bruce McCarl

showing the data for this item. The data may be reordered via the mouse. For example placing the mouse on the column containing san francisco and dragging the column up yields



which is the slice of the matrix containing san francisco data. In turn clicking on Brussels yields



Also note that the columns are sortable in the left hand portion of the display. All one needs to do is to click on the gray boxes (Symbol, Type,...) with the mouse. There is also a search dialog at the bottom that permits one to look for select items.

Finally note a right mouse click allows one to write the contents of any or all items to HTML format.

15.4.3 Identifying contents with Gdxdump

GAMS distributes a utility, Gdxdump, that will write out data in GMS CSV and other formats

- Data for a selected set, parameter, variable or equation (under all three of the output options when a specific item is named using the SYMB option)
- Data for all sets, parameters, variables and equations (Under *normal* option when the SYMB is

© 2022 Prof. Bruce McCarl

not used)

- Data on solution items (variables and equations) formatted in a fashion suitable for import as a basis in another GAMS program where the marginals and levels are output (as discussed in the Expanded GAMS user guide under the heading *Advanced basis formation in GAMS*).all of the scalars, sets and parameters (tables) in a GDX file to standard output formatted as a GAMS program with data statements or in CSV format. It skips information for variables and equations. The syntax is

```
Gdxdump gdxfilename format=choice symb=optional choice
```

where the gdxfilename is the name of the GDX file to write data from. This output is created to the screen not to a file. If one wishes to dump this to a file one uses a command like

```
Gdxdump gdxfilename > filetouse.gms
```

There are some peculiarities in the GDXDUMP. Namely

- Under the format=*CSV* choice it only creates output when a symbol is selected using the SYMB syntax.
- Under the format=*CSV* choice when the requested symbol is a variable or an equation one only gets the level values not the marginal, under the other formats one gets all items.
- Under the format=*gamsbas* choice one gets all variables and equations when the SYMB syntax is **not** used.

One can also write the GDX file contents into a GMS file as in the example below

Further details and additional options are discussed in the document *GDXDUMP*.

Example:

For example when we use the command

```
Gdxdump gdxfilename > filetouse.gms
```

then the contents of filetouse.gms are

```
*  GDX dump of tran2.GDX
*  Library version    : _GAMS_GDX_V224_2002-03-19
*  File version   : _GAMS_GDX_V224_2002-03-19
*  Producer       : GAMS Rev 132  May 25, 2002
*  Symbols        : 8
*  Unique Elements: 5

Set i(*) canning plants/
```

© 2022 Prof. Bruce McCarl

```
  seattle  ,
  san-diego  /;

Set j(*) markets/
  new-york  ,
  chicago  ,
  topeka  /;

Parameter d(*,*) distance in thousands of miles/
  seattle.new-york 25 ,
  seattle.chicago 17 ,
  seattle.topeka 18 ,
  san-diego.new-york 25 ,
  san-diego.chicago 18 ,
  san-diego.topeka 14 /;

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

Parameter dem(*) capacity of plant i in cases/
  seattle 350 ,
  san-diego 600 /;

Parameter sup(*) demand at market j in cases/
  new-york 325 ,
  chicago 300 ,
  topeka 275 /;

* skipped Variable x

* skipped Equation supply
```

where note the variable and equations are skipped at the bottom.

A second Example:

Another example exercising this is (gdxdumpexample.gms)

```
execute 'gams agreste gdx=junk'
execute 'gdxdump junk format=CSV output=filecsv.csv symb=llab'
execute 'gdxdump junk format=CSV output=filecsv2.csv symb=xcrop'
execute 'gdxdump junk format=CSV output=filecsv3.csv symb=landb'
execute 'gdxdump junk format=gamsbas output=filegamsbas.gms symb=xcrop'
execute 'gdxdump junk format=gamsbas output=filegamsbas2.gms'
execute 'gdxdump junk output=filenormal.gms symb=llab'
execute 'gdxdump junk format=normal output=filenormal1.gms symb=xcrop'
execute 'gdxdump junk output=filenormal2.gms symb=landb'
execute 'gdxdump junk output=fileall.gms'
```

© 2022 Prof. Bruce McCarl

15.4.4 Identifying differences in contents with Gdxdiff

GAMS also distributes a utility that looks for differences in two GDX files creating a list of item names that differ and yet another GDX file that exactly specifies the differences.

GDXDIFF compares the data of for items in two GDX files and writes a GDX file showing the differences. In particular for all items with the same name, type and dimension in the two GDX files the differences in numerical values are written to a third GDX file with A summary report written to standard output (ordinarily the LOG file).

This utility is used by using $Call or Execute with the line

gdxdiff file1 file2 {diffile} {Eps = value} {RelEps = value} {Field = FieldName} {ID=Identifier}

GDXDIFF requires the first two file name parameters,

File1	Name of the first GDX file
File2	Name of the second GDX file

The remaining parameters are optional

Diffile	An optional third file name that is the name of the GDX file that contains the differences found in the parameters. If that parameter, is absent the file will be named 'diffile.gdx' and placed in the current directory.
Eps = value	A tolerance that is the maximum amount that two numbers may differ by ie given a1 and a2 then abs(a1-a2) is reported as different if it exceeds this tolerance
RelEps = value	A tolerance that is the maximum percentage amount that two numbers may differ by ie given a1 and a2 then abs(a1-a2)/max(abs(a1),abs(a2)) is reported as different if it exceeds this tolerance. Note this is more complex as discussed in the gdxutilities document under help and tools.
Field = FieldName	A field that if specified limits doen between the information for variables and equations to specific attributes (Lo, L, Up, M, Scale and Prior)
ID=Identifier	Limits the comparisons to selected items; items not specified will be ignored. Multiple items can be specified as: ID=id1 ID=id2 or ID="id1 id2"

More is found on this in the GAMS utilities document at gdxdiff or as found through the IDE help under docs.

© 2022 Prof. Bruce McCarl

Example:

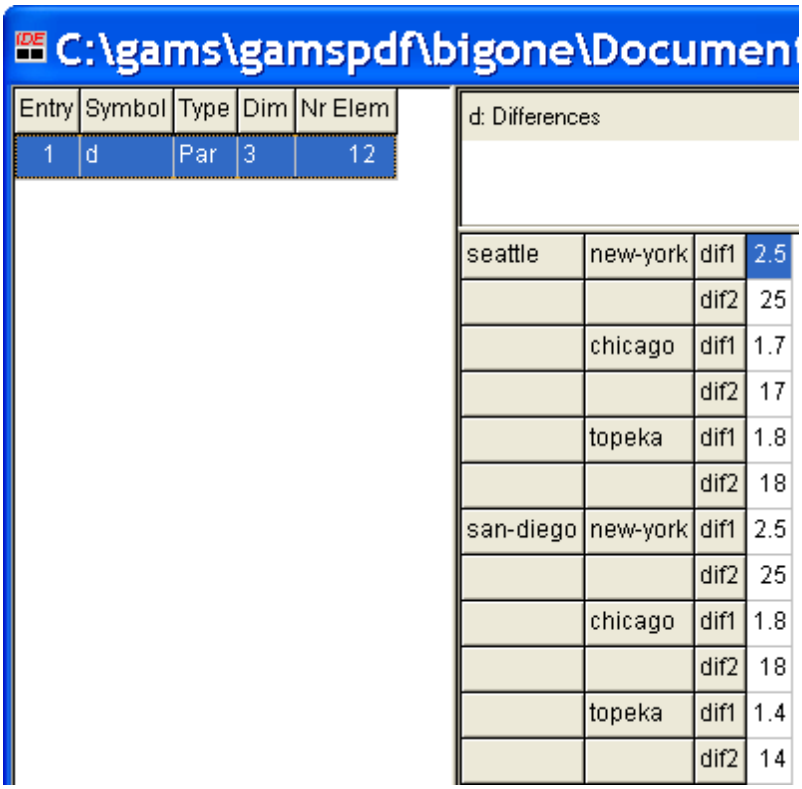
Suppose we wish to compare the GDX files tran and tran2, then we would use the command

```
Gdxdiff tran tran2
```

In turn the output to standard output (nominally the terminal screen) appears as follows

```
Summary of differences:
     d   Data is different
   dem   Keys are different
   sup   Keys are different
supply   Symbol not found in file 1
     x   Symbol not found in file 1
```

and summarizes the differences found. Simultaneously the file **diffile.gdx** when examined in the IDE contains the following



which reports on the differences found in the two files.

Notes:

- Some new coding is introduced in the difference GDX file. Namely a new dimension is added to the parameters being compared which can contain 4 entries

© 2022 Prof. Bruce McCarl

- dif1 indicates that the entry occurs in both files and shows the value found in the first file.
- dif2 indicates that the entry occurs in both files and shows the value found in the second file.
- ins1 indicates that the entry only occurs in the first files and shows the value found.
- ins2 indicates that the entry only occurs in the second file and shows the value found.

- Only named items with the same name, type and dimension will be compared in the **diffile.gdx** output. Named items that are new or are deleted will only appear in the standard output summary report

15.5 Merging GDX files

GDXMERGE is a utility distributed with GAMS that can be used to combine the information from several GDX files into one composite file and **in turn can be used to compare their contents**. Symbols with the same name, dimension and type that appear in the separate files are combined into a single symbol with an added dimension in the first index position that gives the file name.

The procedure is uses as follows :

gdxmerge *filepattern1 filepattern2 …. filepatternn optional-parameters*

optional parameters:

id=ident1, ident2 only merge the symbols ident1, ident2 etc
exclude=ident1, ident2 merge all symbols except ident1, ident2
big=integer the size for big symbols (see below)

where the filepattern entries represent are either individual file names or a wildcard representation using ? and *.

The result will be written to a file called merged.gdx.

In the resultant GDX file one cal find all of the data in the source GDX files plus a set named Merged_set_1 that contains the names of all the files processed during the merge operation. In that set the explanatory text contains the date and time of the gdx file processed.

Notes:

- The file merged.gdx will not be used in a merge operation even if the name matches a file pattern.
- Symbols with dimension 20 cannot be merged, because the resulting symbol will have dimension 21 which exceeds the maximum dimension allowed by GAMS.
- By default, the program reads all gdx files once and stores all data in memory before writing the merged.gdx file. A parameter big=number can be used to specify a cutoff for symbols that will be written one at a time. Each symbol that exceeds the size specified by big will be processed by reading each gdx file and only process the data for that symbol. This can lead to reading the same gdx file many times, but it allows the merging of large data sets. The formula used to calculate the cutoff is: Dimension * TotalNumberOfElements. The number is doubled for variables and equations.

Example (gdxmerge.gms):

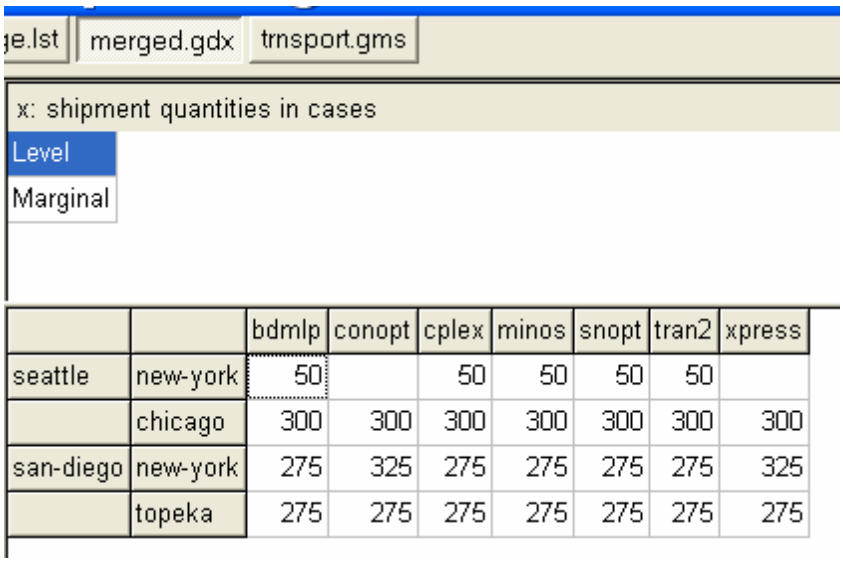
Suppose we solve the trnsport model from the model library using different LP solvers and wish to

© 2022 Prof. Bruce McCarl

compare the runs. To do this After each run, we write all
symbols to a gdx file and then use GDXMERGE to merge the solution information. Then suppose we compare the results or the two dimensional trnsport model solution variable X reading the X from the merged GDX file into an array called ALLX that has the file names as its first dimension

```
$call gamslib trnsport
$call gams trnsport lp=bdmlp gdx=bdmlp
$call gams trnsport lp=cplex gdx=cplex
$call gams trnsport lp=xpress gdx=xpress
$call gams trnsport lp=conopt gdx=conopt
$call gams trnsport lp=minos gdx=minos
$call gams trnsport lp=snopt gdx=snopt
$call gdxmerge bdmlp.gdx cplex.gdx xpress.gdx conopt.gdx
minos.gdx snopt.gdx
set i supply set
    j demand set
    merged_set_1 names of gdx files
variable AllX(merged_set_1,i,j);
*load i and j from one of the solver gdx files
$gdxin bdmlp.gdx
$load i
$load j
*load merged file
$gdxin merged.gdx
$load merged_set_1
$load AllX=X
$gdxin
option AllX:5:1:2;
display i,j,merged_set_1,AllX.L;
```

Instead of using the display statement, we can also use the GAMSIDE to view the merged.gdx file by opening it in the editor and looking at X then dragging the indices around into the order wanted.:



		bdmlp	conopt	cplex	minos	snopt	tran2	xpress
seattle	new-york	50		50	50	50	50	
	chicago	300	300	300	300	300	300	300
san-diego	new-york	275	325	275	275	275	275	325
	topeka	275	275	275	275	275	275	275

© 2022 Prof. Bruce McCarl

15.6 Using GDX files to interface with other programs

The very name GDX – GAMS data exchange suggests this is the mechanism via which users will be able to exchange data with other programs. Today however this usage, while contemplated, is still under development and only exists for selected cases. In particular, there are mechanisms for spreadsheets and a couple of other programs plus a general AOI interface. Let me briefly cover these.

Spreadsheets
Other
API to interface from other programs

15.6.1 Spreadsheets

There are currently three GDX supported pathways for data exchange to spreadsheets

- Rutherford's Xlexport, Xldump and Xlimport
- Gdxxrw
- Gdxviewer

All are discussed in the chapter Links to Other Programs Including Spreadsheets.

15.6.2 GEMPACK

GAMS distributes and supports utilities for converting GEMPACK HAR (header array) files to and from GDX files. These are called **gdx2har** and **har2gdx**. Details about these utilities can be found at http://www.gamsworld.org/mpsge/debreu/gdxhar/index.html.

15.6.3 Other

Utilities for other types of exchanges are now under development as is a general set of procedures for reading and writing GDX files. Users needing to do such exchanges should contact GAMS Development.

15.7 Gdxcopy Making GDX files compatable

GDX files can be incompatible between newer and GAMS versions prior to 22.3 due to compression among other changes. A current GAMS system can read all older GDX file formats. Older file formats may be written using gdxconvert.

The GDXCOPY utility provides a mechanism to convert GDX files to a format that older GAMS systems can read. It is used by employing the syntax

gdxcopy outputoption inputfile outputfiledir

where the items ate

Outputoption Identifies the type of output file to be created

© 2022 Prof. Bruce McCarl

Option	Target format
-V5	Version 5
-V6U	Version 6 uncompressed
-V6C	Version 6 compressed
-V7U	Version 7 uncompressed
-V7C	Version 7 compressed

inputfile Name of the input file to be converted . This file must have an .gdx file extension

outputfiledir Output directory

Note: Version 7 formatted files were introduced with version 22.6 of GAMS; version 6 formatted files were introduced with version 22.3 of GAMS. Prior versions used version 5.

Some features introduced in version 7 of the gdx file format cannot be represented in older formats.

Feature	Action taken
Dimension > 10	Symbol is ignored
Identifier longer than 31 characters	Truncated to 31 characters
Unique element longer than 31 characters	Truncated to 31 characters
Domain of a symbol	Domain is ignored
Aliased symbol	Symbol is entered as a set
Additional text for symbol	Additional text is ignored

Notes:

- The Macintosh Intel-based system (DII) which was introduced with GAMS 22.6 does not support gdx conversion into formats version 6 and version 5.
- The Solaris 10 or higher Intel-based system (SIG) which was introduced with GAMS 22.5 does not support gdx conversion into formats version 5.

© 2022 Prof. Bruce McCarl

- Solaris 9 or higher on Sun Sparc64 (SOX) which was introduced in GAMS 22.6 does not support gdx conversion into formats version 6 and version 5.

15.8 Writing older GDX versions with GDXCONVERT

Over time GDX file formats have changed. Some environments have multiple GAMS versions some of which are older and file format compatibility can become an issue. Consequently one may need to write GDX files in an older GDX file format. This may be done using the environment variable or GAMS parameter gdxconvert and possibly gdxcompress.

Gdx file formats are named v5, v6 or v7 and can be compressed or not compressed.

- v5 applies to GDX files written by GAMS versions 22.2 and earlier.
- v6 were introduced with version 22.3 of GAMS and lasted through 22.5.
- v7 formatted files were introduced with version 22.6 of GAMS.
- GAMS platforms that were introduced after 22.3/22.6 (e.g. Mac Intel or SunSparc64) do not support V5/V6.
- `gdxcompress` allows one to tun on and off compression.

There are two ways of implementing this through environment variables and through a GAMS command line parameter. Command line options have higher precedence than the environment variables with the same name.

16 Links to Other Programs Including Spreadsheets

When a modeler wants to link GAMS results or input to other programs, it can be done in several fundamentally different ways.

- GAMS is in charge and data from other programs is to be incorporated into the GAMS program as it starts up.
- GAMS is in charge and data from the GAMS results are to be passed to other programs at the conclusion of the GAMS run.
- GAMS is in charge and the user wants to run another program during a GAMS run.
- GAMS is in charge and the user wishes to pass data interactively to other programs during a run.

© 2022 Prof. Bruce McCarl

- Equations in the user model are defined by an external program.
- Some other program is in charge and the user wants to use GAMS to solve a model.
- A GAMS model needs to be converted to another language for solution.

The first five of these are discussed herein, the other two are discussed in the chapter Controlling GAMS from External Programs.

Executing an external program
Passing data from GAMS to other programs
Passing data from other programs to GAMS
Customized data interchange links for spreadsheets
Using equations defined by external programs

A wiki is available that provides some information in support of interfacing efforts. see discussion here

16.1 Executing an external program

External programs may be run during a GAMS job either using the $Call, Execute or Put_utility syntax. The $Call procedure is executed at the moment that is encountered during compilation. The Execute and Put_utility commands causes the external program to be run during GAMS program execution. The contrast between these statements is important in two ways.

- Inability to use results from GAMS program execution-- Anything run with $Call can generate files that can be included in the subsequent compilation. On the other hand files generated with Execute and Put_utility cannot be included because $Include operates only at compile time (unless you use Save and Restart). Note there is one exception using a call of GAMS with GAMS as discussed below.
- Ability to feed current results into the external program-- Obviously when one is running an external program there is the desire to pass it data depicting results of the GAMS execution. $Call cannot do this as the data passed have to exist at compile time and cannot use the result of any GAMS calculations and solves in the current program. Execute commands on the other hand can use any data generated during a run which arise before the Execute and Put_utility command's position in the file through passage via put files or other mechanisms.

The big difference between the $Call and Execute is

- $Call
  - can generate results to be immediately incorporated back into GAMS
  - cannot use GAMS results generated within this run because the $Call is executed at compile time.
- Execute and Put_utility
  - can cause a program to be started using results generated by the GAMS program (note such results do have to have been saved in an external file using a command like put)
  - cannot generate results which can be immediately reincluded into the GAMS program because new material can only be added compile time. (Excepting through use of a GAMS from GAMS approach as discussed below.)

© 2022 Prof. Bruce McCarl

$Call
Execute
Put_utility
Timing of execution with $Call and Execute

16.1.1 $Call

The $Call is a dollar command as explained in the Dollar Commands chapter. This command uses the syntax

```
$Call commandtoexecuteinOS
```

or

```
$Call =commandtoexecuteinOS
```

to execute a program or operating system command specified by commandtoexecuteinOS during compilation and before the inclusion of any subsequent include statements. This halts compilation until the operating system indicates that file has been run (Note the = may be needed to make this always be true as explained in the Notes section below). One can create a file in the external program and then use an Include command to bring in that file within the current GAMS program.

Exampes:

(xlimport.gms)

$Call is used in Rutherford's spreadsheet interface Xlimport program. In the current version, two external programs are used. Namely Gdxxrw is used via a $call to get data out of a spreadsheet and Xldump is used also via a $call to read the GDX data and convert it into a GAMS readable format in the file xllink.in then those data are **immediately** included into the GAMS program permitting range checking.

```
$call Gdxxrw "%XLS%" @xllink.txt log=xllink.log

* If filtered import, read from GDX
$if %more% == '' $goto unfiltered
$GDXin xllink.gdx
$load %sym%
$GDXin
$goto term

$label unfiltered
$onempty
$if ParType  %sym% parameter %sym% /
$if SetType  %sym% set %sym% /
$call GDXdump xllink.gdx symb=%sym% noheader > xllink.in
$include xllink.in
```

Notes:

© 2022 Prof. Bruce McCarl

- Generally GAMS will pause until the external job is completed so the next statement after the $Call can use the GAMS include syntax to incorporate any files created by running the external program.
- The other commands in this sequence are explained in the Conditional Compilation or Dollar Commands chapters.
- In some applications the $Call statement needs to include an = before the name of the program called as follows.

```
$call =XLS2gms "i=c:\my documents\test.xls" o=d:\tmp\test.inc
```

This may be required to make GAMS wait until a program is finished. This will be needed for programs that are true windows applications. One can see if this is the case by running a program from the DOS command prompt. If control is returned to the command prompt before the program is finished then the = is needed. A technical explanation of why this is necessary can be found in the **Xls2gms** section of tools documentation.

16.1.1.1 Spaces in file names and paths

One needs to be cautious in the GAMS commands that address external programs and files in dealing with spaces in the file names or paths. Namely these need to be encased in quote as in the following

```
$call =XLS2gms "i=c:\my documents\test.xls" o=d:\tmp\test.inc
```

However even more complex statements are required in the case of Execute where one would use

```
Execute '=XLS2gms "i=c:\my documents\test.xls" o=d:\tmp\test.inc'
```

enclosing the whole thing in single quotes and elements within containing spaces in double quotes.

These practices need to be followed with respect to all places file names can be specified including $Call, $GDXin, $GDXout, Execute_Load, Execute_Unload, Execute_Unloaddi, XLIMport, Gdxxrw etc.

16.1.2 Execute

This command uses the syntax

```
Execute commandtoexecuteinOS
```

or

```
Execute =commandtoexecuteinOS
```

or

```
Execute.ASync commandtoexecuteinOS
```

or

```
Execute.ASyncNC commandtoexecuteinOS
```

© 2022 Prof. Bruce McCarl

to execute a program or OS command specified by commandtoexecuteinOS. The execution occurs during the GAMS program execution.

- The = may be needed to make GAMS wait until the program is done as discussed in the Notes section below.
- The .Async suffix makes GAMS go ahead without waiting.
- The `.ASyncNC` option tells the operating system to start the run a new console rather than sharing the console of the parent process allowing use of multiple processors.
- Since this occurs during execution one cannot use the compile time $Include to incorporate the results of that external run into the GAMS code except through a GAMS from GAMS approach as discussed below or through save and restart use (see the Save Restart chapter).
- One can also use the command SplitOption here is one wants to provisionally change calling parameters for the executable.

Notes:

- The Execute statement may need to include the suffix .ASync to make GAMS proceed without waiting until a program is finished .
- The Execute statement may need to include an = before the name of the program called as follows.

```
Execute '=XLS2gms "i=c:\my documents\test.xls" o=d:\tmp\test.inc'
```

This is required to make GAMS wait until a program is finished. This is needed for programs that are true windows applications. One can see if this is the case by running a program from the DOS command prompt. If control is returned to the command prompt before the program is finished then the = is needed. A technical explanation of why this is necessary can be found in the **Xls2gms** section of tools documentation

Examples:

(gnuplotxyz.gms)

Execute is used in gnuplotxyz to graph data generated within a GAMS program. It accomplishes this by first saving the data to a file using put commands then executing the wgnuplot program which in turn **reads that file**. The Execute command can also involve DOS commands as illustrated below

```
put 'plot ';

loop(%gp_scen%,
  if (gp_count > 1, put ',';);
  file.nw = 0

  put  '\'/' "gnuplot.dat" index ',(gp_count-1):0:0,
       ' using 1:2 "%lf%lf" title "',%gp_scen%.tl,'"';

  file.nw = 6;
```

© 2022 Prof. Bruce McCarl

```
  gp_count = gp_count + 1;
); put /;
$if  "%gp_term%"=="windows" execute 'if exist gnuplot.ini del gnuplot.ini >nul'
$if  "%gp_term%"=="windows" execute 'copy gnuplot.inp gnuplot.ini >nul';
$if  "%gp_term%"=="windows" execute 'wgnuplot';
$if not "%gp_term%"=="windows" execute 'wgnuplot gnuplot.inp';
```

16.1.3 Put_utility

GAMS currently does not allow one to easily manipulate strings however put file commands allow one to write out strings composing them based on set element text. There are cases where when executing external programs or in reading and writing information to external files that such string manipulation can be valuable. For example one might want to read in a large number of GDX files in a similar manner or execute external programs with altering parameters. The put_utility or put_utilities commands in GAMS allow such functions.

In general the put_utility language feature works in the following way. Two lines need to be generated with the put_utility for each command to be executed

- The first line tells what type of put_utility feature to use where the allowed features and their functions are given below.
- The second line gives the arguments to use with that feature
- The general syntax is

```
File nameoffiletouse
Put nameoffiletouse
Put_utility localname 'feature1' / 'arguments' ;
Put_utility 'feature2' / 'arguments' ;
```

or

```
put_utility 'key'/ 'arguments'/ 'key' / 'arguments' /.... ;
```

Notes

The localname is optional and gives the local name of the file that is to be used to pass instructions. In the put_utility command the "/" separates the lines as in ordinary put commands. Note an ending "/" is not used although one may stack features as shown in the example below

The names of the features or keys above that can be used and nature of associated arguments are

Feature name	Nature of General Function	Nature of associated arguments
Exec	Causes a command to be passed to the operating system for execution. GAMS waits for command to finish	Command to be executed with arguments
Exec.ASync	Causes a command to be passed to the operating system for execution but GAMS does not wait for execution to finish	Command to be executed with arguments

© 2022 Prof. Bruce McCarl

Exec.ASyncNC	Causes a command to be passed to the operating system for execution using a different console than the parent process. GAMS does not wait for execution to finish allowing use of multiple processors	Command to be executed with arguments
Shell	Causes a command to be passed to the command shell processor that in turn is passed to operating system for execution	Command to be processed by shell processor then passed on in processed form to the operating system (Note distinctions between shell and exec are technical and can be operating system specific. They typically involve the ability to use redirect of standard input output and the error console)
Gdxin	Causes subsequent GDX file loaded by execute_load or execute_loadpoint to come from a specified file name	Name of the GDX file that data will be loaded from
Gdxout	Causes subsequent GDX file unloaded by execute_unload to come from a specified file name	Name of the GDX file to which data will be unloaded
Ren	Causes put file output to be directed to a named external file	File name to use for subsequent put operations
Inc	Causes the contents of a file to be incorporated into the currently active put file	File name whose contents are to be included
Click	Causes a clickable file reference to be added to the process window	File name for file to which reference will point
Msg	Causes a message to be placed in the listing file	Text of message
Log	Causes a message to be placed in the log file (also process window)	Text of message
Msglog	Causes a message to be placed in both the log and listing files	Text of message

© 2022 Prof. Bruce McCarl

Title	Causes the title on the DOS window to be changed	New name for window
Glb	Used by GAMS to aid in building model library – not intended for users	--
Htm	Used by GAMS to aid in building model library – not intended for users	--
WinMsg	Used to send a message to a Window on windows machine	window name and message/ see example in the test library model ASYNNTRP. There is also an example in the file WindowsReceiver.zip. One can also recieve these messages usinf MessageReceiverWindow as illustrated in the GAMS Test Library model MRW01

The arguments are typically in quotes and limited to 255 characters although in the messages, exec and shell cases multiple quoted elements can be included each up to a maximum of 255 characters and will be included in information passed on.

The advantage in all of these is that you can assemble the command line at run time (e.g. selecting the filename or command line arguments based on some set elements).

Examples (pututility.gms)

Examples

*write stuff to different files

```
loop(i,
  random = uniformint(0,100);
  put_utility 'shell' / 'echo ' random:0:0 ' > ' i.tl:0;
);
```

*Put data in several gdx files then reloads it

```
file fx2;
put fx2;
set ij / 2005*2007 /;
scalar random;
```

*put out the data to multiple GDX files

```
loop(ij,
```

© 2022 Prof. Bruce McCarl

```
          put_utility 'gdxout' / 'data' ij.tl:0;
          random = uniform(0,1);
          execute_unload random;
        );
```

*Load the data from multiple GDX files

```
      loop(ij,
          put_utility 'gdxin' / 'data' ij.tl:0 ;
          execute_load random; display random;
        );

        file dummy; dummy.pw=2000; put dummy;
```

*here I execute some commands with waiting

```
        put_utility 'exec'  / 'gams sets' /
                    'shell' / 'dir *.gms'    ;
```

*here I execute some commands without waiting

```
        put_utility 'exec.async'  / 'gams sets' /
                    'shell' / 'dir *.gms'    ;
```

*here I enter a clickable link

```
        put_utility            'click' / 'sets.html' ;
```

*here I vary where the put file output goes

```
        loop(i,
            put_utility  'ren' /  i.tl:0 '.output'  ;
            put "output to file " i.tl:0 " with suffix output " /;
            );
```

*here i put messages in the LOG and LST files

```
        put_utility 'msg'  / 'message to lst file' /
                    'log'  / 'message to log file' /
                    'msglog'  / 'message to log and lst file' ;
```

*here i put some text in the put file

```
          file junk;
        put junk;
        put_utility  'inc'  / 'addit.txt' ;
        put_utility  'inc'  / 'sets2putuiliity.gms' ;
```

16.1.4 Timing of execution with $Call and Execute

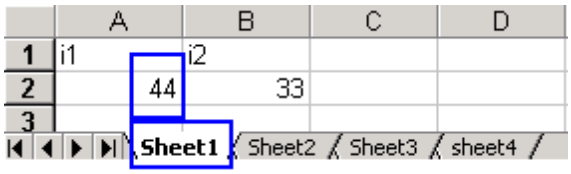
The timing of program execution can at times be confusing. Consider the following example (**callexecute.gms**)

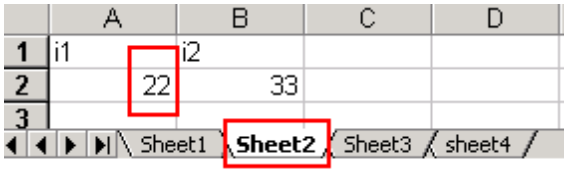
```
set i /i1,i2/
$onmulti
parameter a(i) /i1 22, i2 33/;
$gdxout ss
$unload a
```

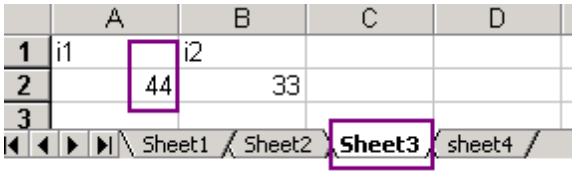
© 2022 Prof. Bruce McCarl

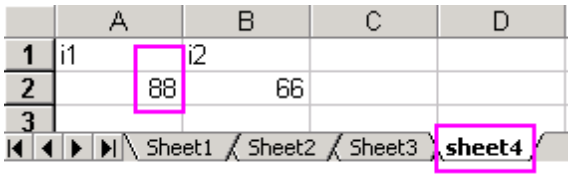
```
$gdxOUT
execute 'Gdxxrw ss.gdx par=a Rng=sheet1!a1'
$Call Gdxxrw ss.gdx par=a Rng=sheet2!a1
parameter a/i1 44/;
a(i)=a(i)*2;
$GDXout ss
$unload a
$Gdxout
$Call Gdxxrw ss.gdx par=a Rng=sheet3!a1
execute_unload 'ss.gdx' , a
execute 'Gdxxrw ss.gdx par=a Rng=sheet4!a1'
```

which uses Gdxxrw as explained below and generates information into four sheets of a spreadsheet workbook as below









Now let me explain the results. First I should note all $ commands and the item redefinition allowed by $Onmulti (which should not ordinarily be used) are resolved at compile time before execution begins. So the statements are implicitly reordered with the $Call and $Unload occurring before the Execute, Execute_Unload.and Put_utility.

- So the $Unload GDX file creation at the bottom occurs before the Execute Gdxxrw at the top and before that I redefined the element a(i1) that appears in cell A2 of each spreadsheet. Thus in sheet1 I have the redefined number for a(i1) (44) that is present at compile time when the Unload GDX file creation at the bottom appears. But note these numbers are unaffected by the execution time a(i)=a(i)*2;

© 2022 Prof. Bruce McCarl

- The $Unload GDX file creation at the top occurs before the $Call Gdxxrw at the top and before that I redefined the element a(i1) that appears in cell A2 of each spreadsheet. Thus in sheet2 I have the original number for a(i1) (22) that is present at compile time when the Unload GDX file creation at the top appears.
- So the $Unload GDX file creation at the bottom occurs before the $Call Gdxxrw at the bottom. Thus in sheet3 I have the redefined number for a(i1) (44) that is present at compile time but note these numbers are unaffected by the execution time a(i)=a(i)*2;
- The Execute Gdxxrw at the bottom follows everything and uses a GDX file created at execution time by the Execute_Unload and thus sheet4 has the final data. This is also true of Put_utility
- Obviously this can be confusing with statements below the one at hand influencing its results. It is never a good idea to intermix Execute, Execute_load, Execute_unload with $Call, $Load and $Unload. I feel all the $ commands should be towards the top and the Executes or Put_utilitys toward the bottom.

16.2 Passing data from GAMS to other programs

GAMS can communicate information to other programs through a variety of mechanisms including

- Put files
- Rutherford's preprogrammed put files
- GDX files
- Specialized links to specific programs like spreadsheets, graphics programs and a few others.

Put file data passage
Rutherford's CSV put: Gams2csv
GDX
Spreadsheet links
Graphics programs
Geographic mapping programs
Gdxviewer links: Access, Excel pivot table, Excel, CSV, GAMS include, HTML, Text files, Plots, XML
Other programs and conversions: Convert, DB2, FLM2GMS, GAMS2TBL, HTML, Latex, MPS, Oracle, XML

A wiki is available that provides some information in support of interfacing efforts. see discussion here

16.2.1 Put file data passage

Put command may be employed to write information to a file that in turn can be read by another program. This may be done using put commands directly or by using Rutherford's canned Gams2csv routine as will be discussed later.

Put files offer a lot of flexibility and are extensively discussed in the chapter Output via Put commands but require programming. Either plain text or CSV delimited files can be generated.

© 2022 Prof. Bruce McCarl

16.2.1.1 Plain text

Plain text put files contain information arrayed in a fixed format for passage to another program. For example, the following segment of code from **regput.gms** writes data in 13 column wide fields from various internally calculated arrays into a file called tosass.put for subsequent use in a statistical program.

```
file tosass;
put tosass;
loop(run,
     put run.tl;
     put @12;
     put /;
     loop(decwant,
          s=  fawelsum( "Agconswelf",decwant,run)/1000;
          put s:13:0;);
     put /;
     loop(decwant,
          s=  fawelsum( "Agprodwelf",decwant,run)/1000;
          put s:13:2;);
     put /;
     loop(decwant,
          s=  fawelsum( "AGtotwelf",decwant,run)/1000;
          put s:13:0;);
     put /;
     );
```

A portion of the resultant file (**tosass.put**) contents is

```
r1
           30           40           50
        60.00        70.00        80.00
         7.56        15.11        22.67
r2
           40           50           60
        70.00        80.00        90.00
         7.56        15.11        22.67
r3
           50           60           70
        80.00        90.00       100.00
         7.56        15.11        22.67
```

The lines gives the case name for the following lines, which are typically the input and output data for a modeled scenario, followed by the data characterizing the case. In turn the target program needs to have instructions prepared that read that data in that format.

© 2022 Prof. Bruce McCarl

16.2.1.2 CSV or otherwise delimited

GAMS programmers may be moving data to external routines that facilitate or require the reading of data in CSV (comma delimited) format (could also involve tab or space delimiters). This may be done by writing a custom set of put file instructions or through Rutherford's Gams2csv as discussed below. When writing a put file, data in CSV format are most simply generated by entering the Put file pc command. This is done in the context of the **regput.gms** example above by adding a single line as follows (**regputcsv.gms**)

```
file tosass;
put tosass;
tosass2.pc=5;
loop(run,
  put run.tl;
  put @12;
put /;
loop(decwant,s=  fawelsum( "Agconswelf",decwant,run)/1000;put s:13:0;);
put /;
loop(decwant,s=  fawelsum( "Agprodwelf",decwant,run)/1000;put s:13:2;);
put /;
loop(decwant,
s=  fawelsum( "AGtotwelf",decwant,run)/1000;put s:13:0;);
put /;),
```

A portion of the resultant file (**tosass2.put**) contents is

```
"r1"
30,40,50
60.00,70.00,80.00
7.56,15.11,22.67
"r2"
40,50,60
70.00,80.00,90.00
7.56,15.11,22.67
"r3"
50,60,70
80.00,90.00,100.00
7.56,15.11,22.67
```

Notes:

- When put commands are run with the pc=5 option then the labels are encased in quotes and the numbers are separated by commas.
- When put commands are run with the pc=5 all the spacing and field widths in the put file will be suppressed.
- Space delimited files can be generated with pc=4 as discussed in the Output via Put Commands chapter.

© 2022 Prof. Bruce McCarl

- Tab delimited files can be generated with pc=6 as discussed in Output via Put Commands chapter.

16.2.2 Rutherford's CSV put: Gams2csv

GAMS users do not have to do put file programming to move data in CSV format. Rather they can use a libinclude routine called Gams2csv developed by Rutherford and associates at the University of Colorado. That program and a write-up is available at *http://www.mpsge.org/gams2csv/gams2csv.htm*.

Gams2csv is invoked as follows:

```
FILE localname /externalname/;
PUT localname;
$LIBInclude gams2csv  [row domain [column domain] ] item[.suffix] [item[.suffi
```

- The material in brackets [ ] is optional
- Before the program can be used a file and initial put command are required as explained in the Output via Put Commands chapter.
- The gams2csv.gms file must have been obtained from the *Rutherford web page* and installed into the place where libinclude files are read from. This location is generally the inclib subdirectory of the GAMS system directory (as discussed in the Including External Files chapter).
- Data from a parameter, variable or equation may be output. When a variable or equation is used, an attribute must be identified such as .L, or .M.
- The routine may be used within a loop or if block only if it is first initialized with a blank invocation (**$LIBInclude gams2csv**).
- All items written in a single libinclude must be of the same dimension. To write items of different dimensions or domains, use multiple libincludes of the routine.
- When row and column domains are specified they can cause less than the full set to be entered into the CSV file. The domain must be the set defining the item or a subset thereof.
- One can control whether zeros are written to the CSV file through the global environment variable "zeros" using the syntax

```
$setglobal zeros yes
```

- One can specify a prefix to be attached to every line using the syntax

```
$setglobal prefix text
```

Example:

(ruthercsv.gms)

```
put 'x without domain' /;
$libinclude gams2csv x
put / 'x with lesser domain for i' //;
$libinclude gams2csv lessi x
```

© 2022 Prof. Bruce McCarl

```
put / 'y without domain' //;
$libinclude gams2csv y
put / 'y with lesser domain for i and j' //;
$libinclude gams2csv lessi,lessj k y
put / 'z without domain' //;
$libinclude gams2csv z
put / 'z with lesser domain for j and k' //;
$libinclude gams2csv i,lessj,lessk z
```

which yields the output

```
x without domain
"x","A one-dimensional vector."
,"i1",1.7174713200000E-01
,"i2",8.4326670800000E-01
x with lesser domain for i
"x","A one-dimensional vector."
,"i1",1.7174713200000E-01
y without domain
"y","A three dimensional array written with column headers"
,,,"k1","k2","k3"
,"i1","j1",3.0113790400000E-01,2.9221211700000E-01,2.2405286700000E-01
,"i1","j2",3.4983050400000E-01,8.5627034700000E-01,6.7113723000000E-02
,"i1","j3",5.0021066900000E-01,9.9811762700000E-01,5.7873337800000E-01
,"i2","j1",9.9113303900000E-01,7.6225046700000E-01,1.3069248300000E-01
,"i2","j2",6.3971875900000E-01,1.5951786400000E-01,2.5008053300000E-01
,"i2","j3",6.6892860900000E-01,4.3535638100000E-01,3.5970026600000E-01
y with lesser domain for i and j
"y","A three dimensional array written with column headers"
,,,"k1","k2","k3"
,"i1","j1",3.0113790400000E-01,2.9221211700000E-01,2.2405286700000E-01
z without domain
"z","A three dimensional array written in list form"
,,,"k1","k2","k3"
,"i1","j1",3.0113790400000E-01,2.9221211700000E-01,2.2405286700000E-01
,"i1","j2",3.4983050400000E-01,8.5627034700000E-01,6.7113723000000E-02
,"i1","j3",5.0021066900000E-01,9.9811762700000E-01,5.7873337800000E-01
,"i2","j1",9.9113303900000E-01,7.6225046700000E-01,1.3069248300000E-01
,"i2","j2",6.3971875900000E-01,1.5951786400000E-01,2.5008053300000E-01
,"i2","j3",6.6892860900000E-01,4.3535638100000E-01,3.5970026600000E-01
z with lesser domain for j and k
"z","A three dimensional array written in list form"
,"i1","j1","k2",2.9221211700000E-01
,"i2","j1","k2",7.6225046700000E-01
,"i3","j1","k2",0.0000000000000E+00
```

16.2.3 GDX

One may also pass information via GDX files as discussed in the Using GAMS Data Exchange or GDX

© 2022 Prof. Bruce McCarl

Files chapter. However, GAMS has not yet released general interface routines for use in custom programs so what one needs to do is write a GDX file then use another GAMS program to write the file passing the data using CSV or other PUT file related procedures as discussed below.

16.2.4 Spreadsheet links

As discussed below one can pass information to spreadsheets using program specific procedures like Xldump, Xlexport, Gdxxrw or Gdxviewer. One can also write CSV files as covered above and import directly or through a spreadsheet importing wizard (for those unfamiliar with such procedures see the coverage in the document **Data Exchange with Excel**.

16.2.5 Graphics programs

Statements may be entered into a GAMS program that permit graphical displays of data computed during a GAMS run directly in a window on a PC using Gnuplot through Gnuplot.gms, gnuplotxyz.gms, Matlab or Excel.

16.2.5.1 Gnuplot

Three procedures have been developed for interface with Gnuplot. These include Rutherford's Gnuplot.gms, Uwe Schneider and my gnuplotxyz.gms and the procedures described in the **Gnuplot**. section of the GAMS User's Guide or here.

The upstart of this is that by inserting a couple of commands in a GAMS program on a Windows machine you can get a graph developed and displayed during any PC GAMS run. It will also work on XWINDOWS under LINUX with a little modification.

16.2.5.1.1 Gnuplot.gms

Tom Rutherford at the University of Colorado originally developed the interface with Gnuplot and distributes it freely on his web site *www.mpsge.org/gnuplot*. I will not illustrate it here as it is invoked in a manner essentially identical to the procedures to invoke gnuplotxyz which are discussed below.

16.2.5.1.2 Gnuplotxyz.gms

Uwe Schneider and I developed a modified version of Rutherford's Gnuplot.gms trying to achieve simpler syntax (containing more default values than Rutherford) and simpler construction of so called (in spreadsheets) XY graphs where the X and Y data are not common across lines in the graph. The package is distributed through *http://www.uni-hamburg.de/Wiss/FB/15/Sustainability/schneider/gnuplot/*.

To graph data in a GAMS program I need to do three basic things to use gnuplotxyz.

- Download the gnuplotxyz software getting both the gms and windows gnuplot executable (wgnuplot.exe).
- Fill a three-dimensional array. In the example **Simplegr.gms** I fill an array named graphdata (you may use any other name) describing the data for the two items to graph where the first dimension is the name of the line, the second gives the set element names of the points to use, and third the data for the point giving the x and y coordinates. Such statements appear

© 2022 Prof. Bruce McCarl

below.

```
LINES      Lines in graph /A,B/
POINTS     Points on line /1*10/
ORDINATES  ORDINATES      /X-AXIS,Y-AXIS/  ;
TABLE GRAPHDATA(LINES,POINTS,ORDINATES)
       X-AXIS   Y-AXIS 0
A.1       1       1
A.2       2       4
A.3       3       9
A.4       5      25
A.5      10     100
B.1       1       2
B.2       3       6
B.3       7      15
B.4      12      36;
```

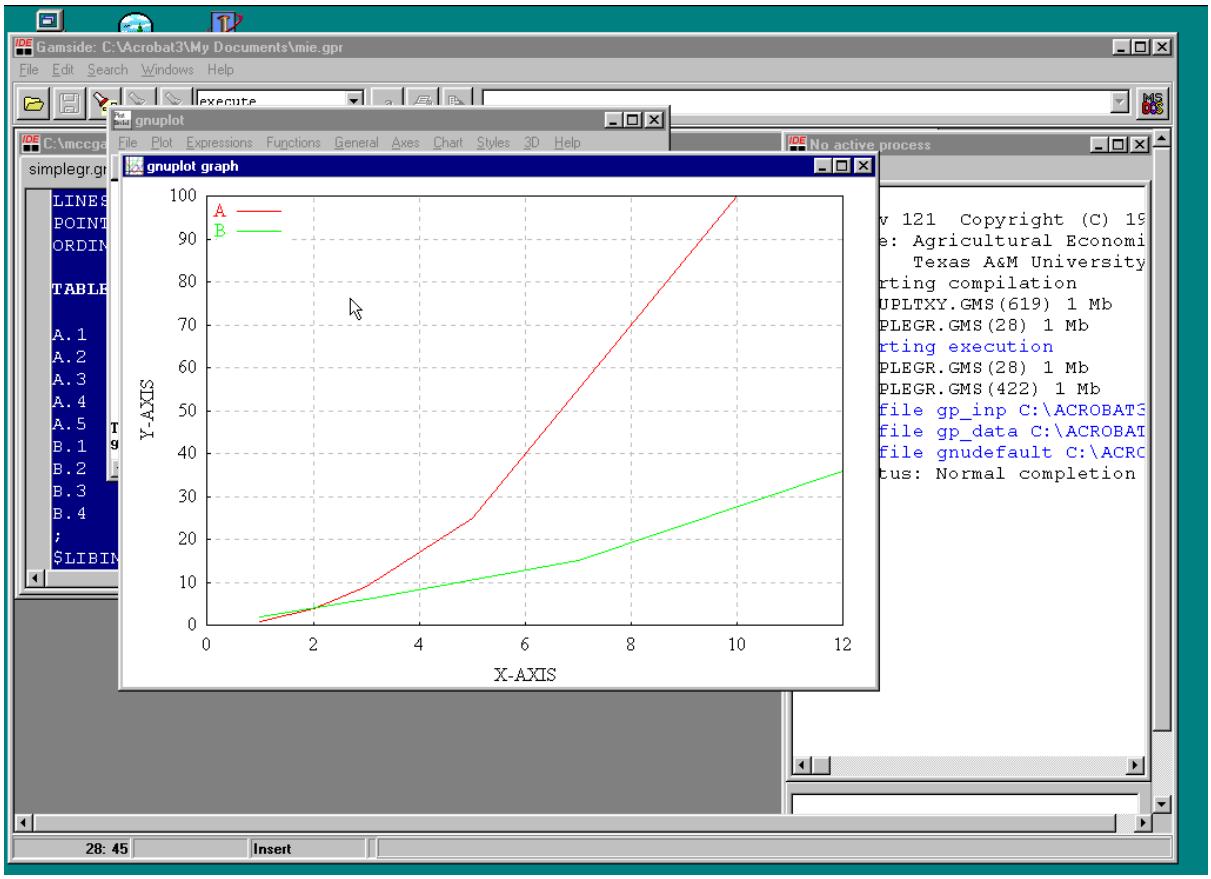
- Then given the data call gnuplotxyz through a libinclude statement

```
$LIBInclude gnuplotxyz GRAPHDATA Y-AXIS X-AXIS
```

where the first argument after gnuplotxyz gives the array name, the second name of a set element in the third array position which contains the data coordinates for the y axis and the third the name of a set element in the third array position which contains the data coordinates for the x axis.

In turn when I run I get two new windows that automatically open in front of the IDE.

© 2022 Prof. Bruce McCarl



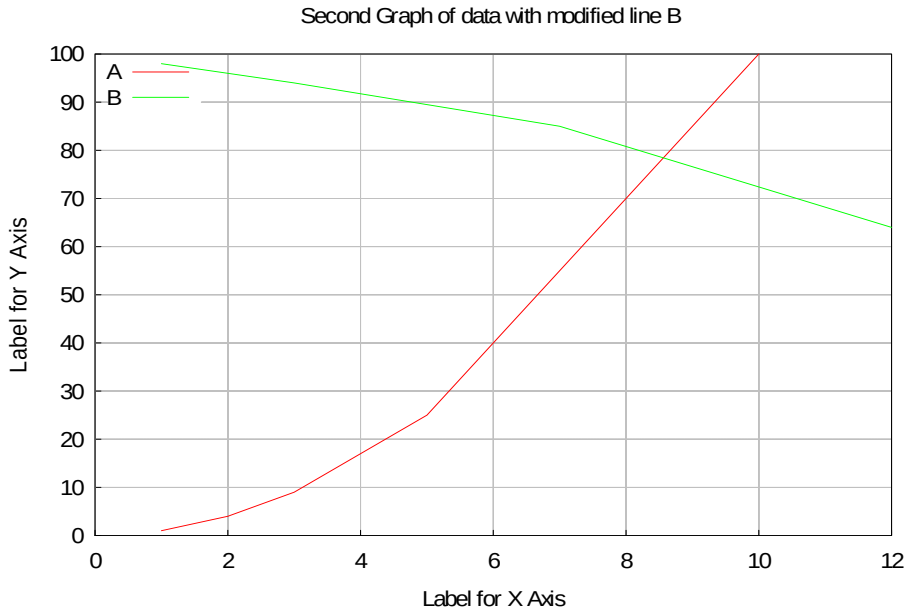
- The window labeled gnuplot graph is the graph of the data and the window labeled gnuplot is a result of the execution of the wgnuplot executable.
- I can plot more than one line by including the libinclude command more than once.
- I can manipulate labels and set a number of options using setglobal commands as discussed in the gnuplotxyz *http://www.uni-hamburg.de/Wiss/FB/15/Sustainability/schneider/gnuplot/* and as illustrated in the Conditional Compilation chapter. In particular, I draw two graphs and manipulate labeling in a manner as illustrated below (**simplegr2.gms**)

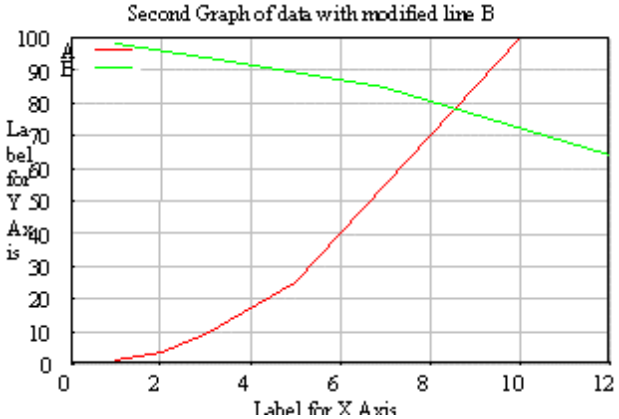
```
$setglobal gp_title  "First Graph of data "
$setglobal gp_xlabel "Label for X Axis"
$setglobal gp_ylabel "Label for Y Axis"
$LIBInclude gnuplotxyz GRAPHDATA Y-AXIS X-AXIS
GRAPHDATA("B",POINTS,"Y-axis")
      $GRAPHDATA("B",POINTS,"Y-axis")=
    100-GRAPHDATA("B",POINTS,"Y-axis");
$setglobal gp_title  "Second Graph of data with modified line B "
$LIBInclude gnuplotxyz GRAPHDATA Y-AXIS X-AXIS
```

where the setglobal commands enter labels for axes and graph title

This yields both the graph above and the graph below generated in 2 windows with 4 total windows generated.

© 2022 Prof. Bruce McCarl





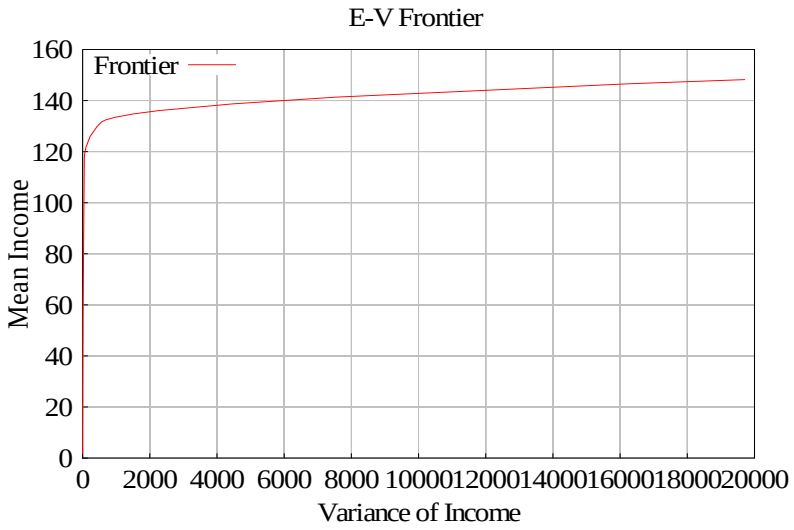
- Many more options are possible as listed in the documentation. Many are embedded in the file **plotopts.gms**
- Calculated data from a model solution can be graphed as illustrated in the example **evportfo.gms**

```
LOOP (RAPS,RAP=RISKAVER(RAPS);
      SOLVE EVPORTFOL USING NLP MAXIMIZING OBJ ;
      VAR = SUM(STOCK, SUM(STOCKS,
INVEST.L(STOCK)*COVAR(STOCK,STOCKS)*INVEST.L(STOCKS))) ;
      OUTPUT("RAP",RAPS)=RAP;
      OUTPUT(STOCKS,RAPS)=INVEST.L(STOCKS);
      OUTPUT("OBJ",RAPS)=OBJ.L;
      OUTPUT("MEAN",RAPS)=SUM(STOCKS, MEAN(STOCKS) *
INVEST.L(STOCKS));
      OUTPUT("VAR",RAPS) = VAR;
```

© 2022 Prof. Bruce McCarl

```
      OUTPUT("STD",RAPS)=SQRT(VAR);
      OUTPUT("SHADPRICE",RAPS)=INVESTAV.M;
      OUTPUT("IDLE",RAPS)=FUNDS-INVESTAV.L   );
parameter graphit (*,raps,*);
graphit("Frontier",raps,"Mean")=OUTPUT("MEAN",RAPS);
graphit("frontier",raps,"Var")=OUTPUT("std",RAPS)**2;
*$include gnu_opt.gms
* titles
$setglobal gp_title  "E-V Frontier "
$setglobal gp_xlabel "Variance of Income"
$setglobal gp_ylabel "Mean Income"
$libinclude gnuplotxyz graphit mean var
```

Here a model is repeatedly solved and a three dimensional array called graphit is built which contains the name of the line to be graphed (frontier) and the mean and variances. These means and variances were calculated using report writing statements into the array called output as discussed in the chapters on Improving Output via Report Writing and Doing a Comparative Analysis with GAMS. The resultant graph is



- Once a graph is in the window, a left click on it makes it available for cut and paste.
- When using gnuplotxyz several things need to be present
  - The gnuplot executable needs to be in the GAMS system directory (nominally c:\program files\gams22.7\). It is downloadable through my website and is called wgnuplot.exe
  - The gnuplotxyz.gms file must be in the inclib directory under the GAMS system directory = (nominally c:\program files\gams22.7\inclib\).
- Sometimes with Windows 2000 when you graph multiple graphs you have to close the first one before the second one becomes visible.

© 2022 Prof. Bruce McCarl

16.2.5.2 Matlab

Matlab can be used to graph GAMS data in much the same fashion as above. The document *GDXMRW: Interfacing GAMS and MATLAB* by Michael C. Ferris, Rishabh Jain and Steven Dirkse explains the Matlab interface.

16.2.5.3 Spreadsheet graphics

One may also use spreadsheet-based graphics as will be illustrated in the spreadsheet section below.

16.2.6 Geographic mapping programs

Some modelers wish to express things in the form of a map depicting geographic movements of goods, production etc. Some mapping programs like *MAPVIEWER* from Golden Software use CSV input formats and can be directly used with the techniques above once the map has been set up. GAMS corporation has also released a library include file called GAMSMAP which uses MAPINFO's PROVIEWER mapping system as described on the GAMS web page at *Exporting to MapInfo Maps*.

Tom Rutherford has also developed a GAMS interface to Mark Horridge's ShapeMap a tool for shading or coloring regions of simple maps and this has been extended by Uwe Schneider and called GAMS-SHADEMAP.

16.2.7 GDX2ACCESS

GDX2ACCESS is a tool that dumps the entire contents of a GDX file into an MS Access file (.mdb or .accdb file).
Every identifier(set, parameter, variable, equation etc) gets its own table in the database.

GDX2ACCESS is resident in GAMS and documented at gdx2access

16.2.8 Gdx2sqlite

GDX2SQLITE is a tool that dumps select items in a GDX file into a SQLITE file (.db file).

GDX2SQLITE is resident in GAMS is documented further here.

16.2.9 Gdxrrw

GDXRRW is an interface to pass data through GDX files to programs written in the R language.

16.2.10 Gdxviewer links: Access, Excel pivot table, Excel, CSV, GAMS include, HTML, Text files, Plots, XML

A program called Gdxviewer will move data from a GDX file and place it a number of places including an Access database. Gdxviewer was developed by Erwin Kalvelagen, at the GAMS Development Corporation and is part of the GAMS Windows distribution. The GAMS IDE contains a GDX file viewer which is accessed when one opens a GDX file.

When run, Gdxviewer organizes the symbols in a tree format on the left hand side of the screen. Simultaneously in the right hand pane, the data for the item selected are shown in tabular format. However, the display column headers are not very descriptive with names like dim1, dim2, etc. because the GDX file does not contain information on domains of parameters and other items. Once one selects an item in the left hand window one can right click on it with the mouse and choose among a number of

© 2022 Prof. Bruce McCarl

export possibilities. Data for the selected object can be plotted or sent to a

- Text file: a plain ASCII file for inclusion into text editors and by programs you write.
- CSV file: a comma separated value files.
- Excel XLS file: a new XLS file ('new workbook') or to a new sheet in an existing spreadsheet. This option works only if Excel is available on your PC.
- Excel Pivot table. An Excel pivot table within a workbook. This only works for parameters or variables/equations with 2 dimensions or more. This option works only if Excel is installed on your PC.
- GAMS include file: for subsequent use through a $Include.
- Access database. A new table will be generated. If the .MDB file does not exist already it will be created. This option works only if Access is installed on your PC.
- HTML file: Data can be exported to HyperText Markup Language: the language of Web pages. Rutherford's GAMS2TBL can also write such files.
- XML file: Data can be exported to XML a web related format to store data.

For more details see the write-up in the gdxviewer.pdf document in the docs\tools sub-directory of your GAMS system directory.

16.2.11 Other programs and conversions: Convert, DB2, FLM2GMS, GAMS2TBL, HTML, Latex, MPS, Oracle, XML

Specialized interfaces exist that allow linkage with a number of other programs.

- The Data and Model Exchange with Other Applications discusses interfaces with Excel, Oracle, Access, DB2, general SQL using procedures, Latex, Gnuplot, HTML, XML and MPS.
- The Application Programming Interfaces to GAMS discusses interfaces with programs in .NET, Java, Python, Fortran, Visual Basic, Delphi, C, and C++.
- Leo Lopes contributed the software program fml2gms.exe that is in the GAMS distribution which converts a CoinFML style XML file into a GAMS formatted file. Running it in a DOS window without parameters gives a rudimentary documentation.
- Thomas Rutherford distributes a utility called *GAMS2TBL* that can write HTML or Latex tables.
- Use of the CONVERT solver allows one to change GAMS files to a number of other formats including AlphaECP, AMPL, BARON, CoinFML, CplexLP, CplexMPS, Dict, FixedMPS, GAMS Scalar format, LAGO, LGO, LINGO, MINOPT or ViennaDag as further discussed in the CONVERT solver guide or the Model Types and Solvers Chapter.

16.3 Passing data from other programs to GAMS

GAMS can retrieve or import information from other programs through

- The include file mechanism,
- GDX files or

© 2022 Prof. Bruce McCarl

- Specialized links to spreadsheets, and a few other programs.

Including data
Spreadsheet links
GDX
Mdb2gms
SQL: Sql2gms
Other programs: DB2, Latex, GNETGEN, Gnuplot, Matlab, MPS, NETGEN, Oracle

A wiki is available that provides some information in support of interfacing efforts. see discussion here

16.3.1 Including data

GAMS may include external files that have been written by other programs providing they exist before the GAMS run instruction is issued, although a trick may be used to get around this as discussed below. File inclusion may be done using the $include, $batinclude or $libinclude syntaxes as discussed in the Including External Files chapter. There are also special provisions regarding inclusion of comma-delimited -CSV- files where such files may be incorporated by using the command $ondelim before beginning the entry and then $offdelim afterwards. File inclusion is extensively discussed in the Including External Files chapter.

16.3.2 Spreadsheet links

Spreadsheet data may be imported into GAMS via three procedures. As discussed below one can use procedures that can either read or write spreadsheet data like Xlimport or Gdxxrw. In addition there is a Windows oriented program Xls2gms that allows you to extract data from an Excel spreadsheet and convert it into a GAMS include file and a program Xldata that reads from Excel files on machines that do not have Excel installed on them. The read/write Xlimport and Gdxxrw procedures are discussed below. Here I discuss Xls2gms and XLSDUMP as they can only import data.

One may also deliver all data in a spreadsheet into a GDX file using XLSDUMP.

16.3.2.1 Xls2gms

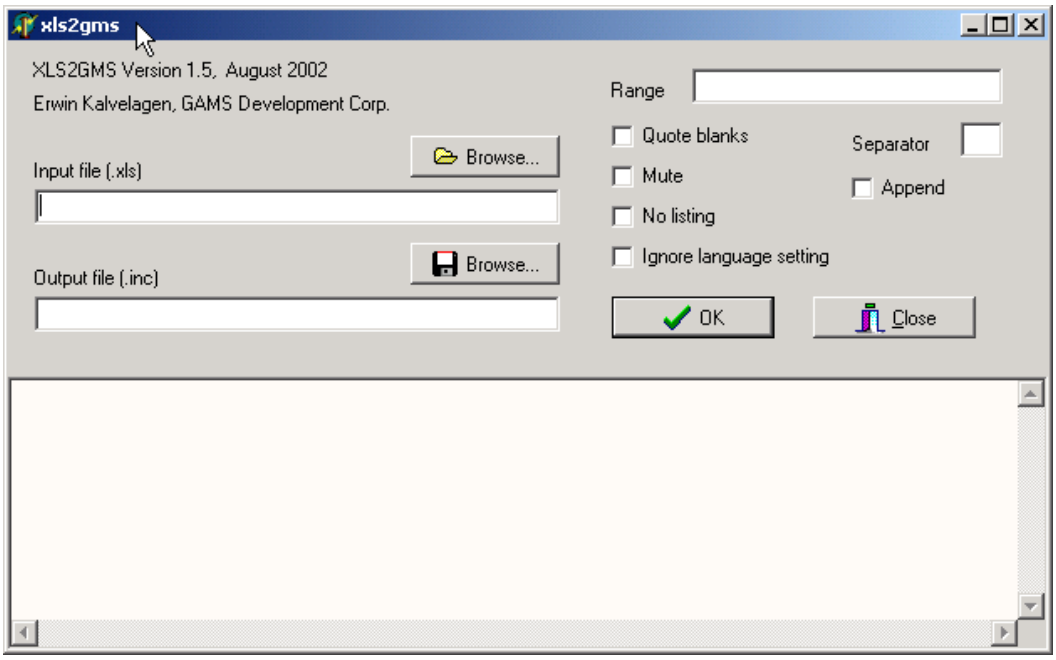
Xls2gms is a utility developed by Erwin Kalvelagen, GAMS Development Corporation that extracts data or GAMS instructions from an Excel spreadsheet and converts it into a GAMS include file. Xls2gms is resident in GAMS and is documented in xls2gms

Xls2gms runs in two different modes – an interactive based windows mode and a batch mode.

16.3.2.1.1 Interactive mode

When Xls2gms.exe is executed without arguments one receives the screen

© 2022 Prof. Bruce McCarl



In turn, if one specifies use of the spreadsheet **tran.xls**, and the range sheet1!a1:d3 one addresses the spreadsheet area below

	A	B	C	D	E
1		new-york	chicago	topeka	
2	seattle	25	17	18	
3	san-diego	25	18	14	
4					

and the program generates an include file as follows

```
* ----------------------------------------------------
* Xls2gms Version 1.5,  August 2002
* Erwin Kalvelagen, GAMS Development Corp.
* ----------------------------------------------------
* Application: Microsoft Excel
* Version:     9.0
* Workbook:    C:\gams\gamspdf\bigone\tran.xls
* Sheet:       Sheet1
* Range:       $A$1:$D$3
* ----------------------------------------------------
           new-york chicago topeka
seattle    25       17      18
san-diego  25       18      14
*----------------------------------------------------
```

This file in turn can be included into GAMS at compile time or at execution time via GDX and the GAMS to GAMS trick discussed below.

© 2022 Prof. Bruce McCarl

Note while the data are output in Table format, the table statement is not present. Thus the user needs to supply it either in the GAMS file or it can be typed above the table in the spreadsheet and Xls2gms will copy it. In particular Xls2gms copies all entries longer than 30 columns verbatim into the GMS file (see the **resource.xls** example as imported by **Xl2gmsresource.gms** as discussed below).

There are a number of options in Xls2gms that merit mention

- Checking the **quote blanks** box causes the program to encapsulate strings that contains blanks in quotation marks and also deals with strings containing quotation marks.
- The box to the right of **separator** allows the user to enter a character that will be used to separate cells in the output file. By default this is a blank. CSV files can be generated by entering a , (comma) in this field.
- Checking the **mute** box will reduce the amount of information written to the output file. The output file then becomes

```
            new-york chicago topeka
seattle     25       17      18
san-diego   25       18      14
```

- Checking the **No listing** box causes the program to enter $offlisting and $onlisting into the program suppressing the data from the LST file.
- Checking the **Ignore language settings** box causes the program to convert numbers in the common European format (where a comma is used for the decimal point --3.14 is written as 3,14) into the US notation using a decimal point employed in GAMS.

16.3.2.1.2 Batch mode

Xls2gms can operate in batch mode. The basic call is of the form

```
XLS2gms i=inputsheet o=outputinclude r=range
```

from a DOS prompt or

```
$call =XLS2gms i=inputsheet o=outputinclude r=range
```

from within GAMS.

In these cases

- The = is needed to make GAMS wait as discussed above.
- Inputsheet is the name of the .xls spreadsheet file.
- Outputinclude is the name of the file where the include file is to be saved.
- One needs to use quotes for file names and paths with spaces in them as discussed above.
- Range identifies the portion of the sheet to import.
  - If range is not specified Xls2gms uses the entire contents of the first sheet in the

© 2022 Prof. Bruce McCarl

workbook.

- If no sheet name is specified (i.e. r=a1:b3) then one gets the given range from the first sheet in the workbook.
- If no range is specified but a sheet is (i.e. r=sheet1!) then one gets the whole sheet.
- Range names are allowed (i.e. r=myrangename).

- Several other parameters may be used
  - @filename identifies a file from which to read command line options where within that file each line contains only one option, typed just as if it were specified on the command line.
  - B causes fields with blanks to be quoted as discussed above.
  - S= specifies a field separator (i.e. s=, yields CSV files) as discussed above.
  - M turns on mute mode as discussed above.
  - L inserts $offlisting and $onlisting as discussed above.
  - P converts numbers to decimal point format as discussed above.

16.3.2.1.2.1 GAMS program in Excel sheet

XL2GAMS may also be used to import GAMS instructions. One can maintain a GAMS program in the spreadsheet and then simply include it in a GAMS program. For example, in **resource.xls** I imbed **resource.gms** (excepting I spaced out the table as one might normally do in Excel in a24:b33). Then I include and execute it using the following GAMS program (**Xl2gmsresource.gms**)

```
$setglobal path "c:\gams\gamspdf\bigone\"
$call =XLS2gms i=%path%resource.xls o=%path%resourceXls2gms.inc
$include %path%resourceXls2gms.inc
```

It appears as if one needs to specify the full path in such a use.

16.3.2.2 XLSDUMP

One may also deliver all data in a spreasdheet into a GDX file using XLSDUMP.

it is run with the syntax

XLSDump *infile outfile*

where:

infile is the name of an Excel workbook

outfile is optional and gives the name of the output gdx file.

If no output file name is specified, the name of the input file will be the root of the input file name plus gdx.

Unlike GDXXRW, the program does not require that Excel is installed.

© 2022 Prof. Bruce McCarl

This works only on windows platforms.

16.3.3 Database links

GAMS can link to Access and SQL databases.

Access is interfaced with through GDX2ACCESS, MDB2GMS. and GDXviewer

SQLite is interfaced with through GXS2Sqlite

SQL is interfaced with through SQL2GMS

16.3.3.1 SQL: Sql2gms

Data may be transferred to SQL compatible databases using Sql2gms - a utility developed by GAMS Development Corporation. Sql2gms extracts data from any SQL relational database via use of the Windows ADO or ActiveX Data Objects procedures. In turn it converts the results into a GAMS include file. Sql2gms is part of the GAMS distribution for Windows and is documented in sql2gms.pdf in the docs\tools directory of the GAMS system directory.

Its function is similar to the Mdb2gms procedure discussed above but with a more complex syntax that also identifies the database manager to use.

16.3.3.1.1 GDX

One may also pass information via GDX files as discussed in the Using GAMS Data Exchange or GDX Files chapter. However, GAMS has not yet released general interface routines rateher having particular ones to MATLAB, ACCESS and Excel so what one needs to do is write a file for GAMS inclusion then include that file in another GAMS program and create a GDX file for use in the current file as discussed below.

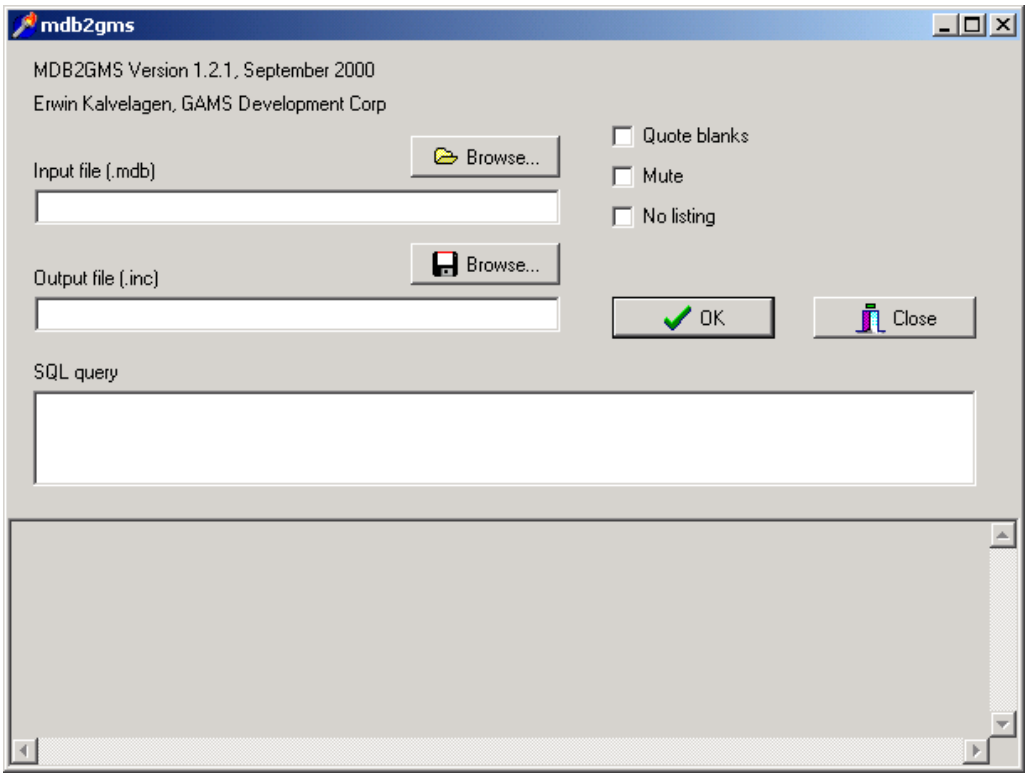
16.3.3.2 Mdb2gms

Mdb2gms is a utility developed by Erwin Kalvelagen, at the GAMS Development Corporation that extracts data from an Access database and converts it into a GAMS include file. Mdb2gms is part of the GAMS distribution for Windows and is documented in mdb2gms/index.html in the docs/tools in the GAMS system directory. The utility runs in two different modes – an interactive based windows mode and a batch mode.

16.3.3.2.1 Interactive mode

When Mdb2gms.exe is executed without arguments then one receives the screen

© 2022 Prof. Bruce McCarl



In turn, if one specifies use of the database **transport.mdb**, and the query

```
select loca,locb,distance from dist
```

one addresses a data table in Access as follows

dist : Table

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

and the program generates an include file as follows

```
* ------------------------------------------------
* Mdb2gms Version 1.2.1, September 2000
* Erwin Kalvelagen, GAMS Development Corp
* ------------------------------------------------
* DAO version: 3.6
```

© 2022 Prof. Bruce McCarl

```
* Jet version: 4.0
* Database:    c:\gams\gamspdf\bigone\transport.mdb
* Query:       select loca,locb,distance from dist
* ----------------------------------------------------
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
```

where in this case the data would need to be input into a parameter with beginning and trailing /'s as discussed in the Data Entry chapter.

There are a number of options in Xls2gms that merit mention

- Checking the **quote blanks** box causes the program to encapsulate strings that contains blanks in quotation marks and also deals with strings containing quotation marks.
- Checking the **mute** box will reduce the amount of information written to the output file. The output file then becomes

```
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
```

- Checking the **No listing** box causes the program to enter $offlisting and $onlisting into the program suppressing the data from the LST file.

16.3.3.2.2 Batch Mode

Mdb2gms can operate in batch mode. The basic call is of the form

```
mdb2gms i=Inputdb o=Outputinclude q=Query
```

from a DOS prompt or

```
$call =mdb2gms i=Inputdb o=Outputinclude q=Query
```

from within GAMS where

- The = is needed to make GAMS wait as discussed above.
- Inputdb is the name of the .MDB Access database file.
- Outputinclude is the name of the file where the include file is to be saved.
- One needs to use quotes for file names and paths with spaces in them as discussed above.

© 2022 Prof. Bruce McCarl

- Query identifies the database SQL query to use.
  - Queries contain spaces and thus have to be surrounded by double quotes.
  - The syntax of the SQL queries is not covered here. Users should refer to the Microsoft Access documentation.
  - When field names or table names contain blanks, they can be specified in square brackets.
  - Query examples

```
Q="select * from mytable"
Q="select year, production from [production table]"
Q="select [GAMS City],value from [example table],CityMapper where [Access
```

- Several other parameters may be used
  - @filename identifies a file from which to read command line options where within that file each line contains only one option, typed just as if it were specified on the command line.
  - B causes the program to enclose fields with blanks in quotes as discussed above.
  - M turns on mute mode as discussed above.
  - L inserts $offlisting and $onlisting as discussed above.

16.3.4 Gdxrrw

GDXRRW is an interface to pass data from programs written in the R language to GAMS through GDX files.

16.4 API usage

GAMS offers a low level application programming interface (API) to various programming languages. Recently GAMS has introduced an object oriented (OO) API that allows to control GAMS from within different programming languages like C#, Java and Python.

Tutorials for these are included in the GAMS system directory on your computer under the subdirectory `<gams system directory>/docs/API`. This document can also be accessed through the GAMSIDE Help menu under the subdirectory Docs in the subdirectory API.

API usage etc is also discussed here.

16.5 Other programs: DB2, Latex, GNETGEN, Gnuplot, Matlab, MPS, NETGEN, Oracle

Program or data input style specific interfaces exist with a number of other programs.

- The Data and Model Exchange with Other Applications discusses interfaces with Excel,

© 2022 Prof. Bruce McCarl

Oracle, Access, DB2, general SQL using procedures, Latex, Gnuplot, HTML, XML and MPS.

- The Application Programming Interfaces to GAMS discusses interfaces with programs in .NET, Java, Python, Fortran, Visual Basic, Delphi, C, and C++.
- The GAMS Matlab interface is described in "Matlab and GAMS: Interfacing Optimization and Visualization Software" at the GAMS Support Wiki .

16.6 Customized data interchange links for spreadsheets

Spreadsheet based data interchange procedures are available for inclusion in GAMS programs that in turn interface with Microsoft Excel. They are designed to take a GAMS data item and place it in a spreadsheet or retrieve data for a GAMS item from a spreadsheet. There are four types of interchange procedures available

- Those based on Rutherford's utilities Xlexport, Xldump, Xlimport
- Those utilizing GDX files through the GAMS Gdxxrw program.
- Use of XLS2GAMS
- Use of XLSDUMP

Each is discussed.

Xlexport, Xldump, Xlimport
Gdxxrw
XLS2GAMS
XLSDUMP
Spreadsheet graphics
Interactively including results

16.6.1 Xlexport, Xldump, Xlimport

Tom *Rutherford* and associates at the University of Colorado developed a set of workbook data exchange utilities in the late 1990's. These were redeveloped in 2002 in cooperation with the GAMS Corporation to utilize GDX files (internally they employ the Gdxxrw procedure that I discuss below). Briefly, the interface routines do the following:

Xlimport reads data from spreadsheets **at compile time**. It can be used to retrieve **preexisting** data from an Excel workbook.

Xldump writes data and element labels into an existing Excel workbook. Row and column order will follow the internal GAMS order as discussed in the Rules for Item Capitalization and Ordering chapter.

Xlexport writes data into spreadsheets according to the row and column labels in the specified target range. The program will transfer data only for the labels are present in the spreadsheet.

The basic usage of the routines involves use of a libinclude command as follows

```
$LIBInclude Xlimport Gamsitem Workbookfile Sheetname!Range
```

© 2022 Prof. Bruce McCarl

```
$LIBInclude Xlexport Gamsitem Workbookfile Sheetname!Range /m
$LIBInclude Xldump Gamsitem Workbookfile Sheetname!Range
```

where

Xlexport, Xlexport and Xldump are names of GMS files resident in the inclib directory of the GAMS system directory (which are automatically placed there when a GAMS release after version 21.0 is installed).

Gamsitem is the name of a parameter, set, variable or equation in the source GAMS program. Attributes are typically used with variables and equations as discussed in the Variables, Equations, Models and Solves chapter.

Workbookfile is the name of an Excel workbook file nominally without the XLS extension where one needs to use quotes for file names and paths with spaces in them as discussed above.

Sheetname!Range is the standard Excel terminology for specification of a set of cells in a workbook.

/m identifies when Xlexport is being used whether data put into the spreadsheet are to be merged or replaced. If /m is specified then the program does not erase existing entries and where a zero is present in an array to be placed in the spreadsheet it will not overwrite an existing nonzero. On the other hand if /m is not specified (left off) then the entire writing range will be cleared in the spreadsheet before any data are written.

16.6.1.1 Xlimport

The procedure Xlimport imports data from a spreadsheet into a GAMS program. Its format is that above requiring specification of the

- Name of the item to have data imported into it.
- The source spreadsheet.
- The location in the sheet including source sheet name and cell range.

Example:

(fromExcel.gms)

```
set places /newyork,chicago,topeka,totalsupply/
    destinaton(places)
    sources /seattle,sandiego,totalneed/
    source(sources);
destinaton(places)=yes;
destinaton("totalsupply")=no;
source(sources)=yes;
source("totalneed")=no;
parameter trandata (sources,places) transport data from spreadsheet
          Supply(Sources)   Supply at each source plant in cases
          Need(places)      Amount neeeded at each market destination in cases;
$libinclude xlimport trandata myspread.xls input!a1:e4
```

This copies the data for the GAMS parameter named trandata from the spreadsheet called **myspread.xls** from the sheet called input from the range a1 to e4.

© 2022 Prof. Bruce McCarl

Notes:

- The use of Xlimport permits simplicity of data transfer, but is not as versatile as the use of Gdxxrw as described below.
- Xlimport is designed to allow one to import set names but this only works under a row vector of set element names. Users wishing to import set element names from the spreadsheet should use Gdxxrw.
- The use of Xlimport to import data requires that the set elements have already been specified in explicit set statements.
- Xlimport is restricted to compile time imports only. Data imports during execution time must use Gdxxrw or the GAMS from GAMS procedure discussed below.
- Xlimport being a compile time import does domain checking to make sure the set element names in the range match.
- Current procedures do not allow import of attributes of variables or equations but this may be fixed by the time this manual is made available.

16.6.1.2 Xlexport

The procedure Xlexport can be used to **export data from a GAMS program into a spreadsheet**. It writes data into spreadsheets only when it finds row and column labels that match the set elements within the specified target range. Those elements also control element ordering. Command format is that above requiring specification of

- Name of the item in the GAMS program to have data exported from it,
- The target spreadsheet,
- The location in the target sheet including the sheet name and cell range.

There is also an optional merge parameter telling whether to leave preexisting data with nonmatching data elements alone.

Example:

(fromExcel.gms)

```
Solve tranport using lp minimizing totalcost ;
$libinclude xlexport transport.l myspread.xls output3!a1..d4
$libinclude xlexport transport.m myspread.xls output3!f1..i4
$libinclude xlexport transport.l myspread.xls output3!a6:d8  /m
$libinclude xlexport transport.l myspread.xls output3!f6:i9  /m
```

This copies the data for the solution levels (.l) or marginals (.m) of the variable named transport into the spreadsheet called **myspread.xls** into various ranges depending on the statement. The first command exports the data after the solve statement into the sheet called output3 in the range a1 to d4. The third command exports the data after the solve statement into the sheet called output3 in the range a6 to d8 merging in the results.

The output3 sheet before the import looks like

© 2022 Prof. Bruce McCarl

myspread.XLS

	A	B	C	D	E	F	G	H
1		newyork	chicago	topeka			chicago	topeka
2	seattle					seattle		
3	sandiego					sandiego		
4								
5								
6		topeka	chicago	random			topeka	chicago
7	sandiego			99		sandiego		
8	seattle					seattle		
9								
10								

output3 / output2 / output / input

and the sheet afterward looks like

myspread.XLS

	A	B	C	D	E	F	G	H
1		newyork	chicago	topeka			chicago	topeka
2	seattle	275	75			seattle		36
3	sandiego	50		250		sandiego	9	
4								
5								
6		topeka	chicago	random			topeka	chicago
7	sandiego	250		99		sandiego	250	
8	seattle		75			seattle		75
9								
10								

output3 / output2 / output / input

where the order of the results varies according the order in which the column labels and row labels appear in the spreadsheet. Also note under the merge option that the data present in the spreadsheet before the export remain afterward when the labels do not match as in the case of the random column in column D rows 6-7. The exports other than the first one do not fully match all of the column dimensionality of the transport.l item and thus only export subsets of the items.

Notes:

- The use of Xlexport either requires the spreadsheet to be closed or it to be shared through the Excel Tools Share Workbook option. However, if shared, the data put by GAMS will only be **reflected in the workbook if you do a file save** and the procedure may be very slow. Often the best option is to close the workbook.
- The use of Xlexport permits simplicity of data transfer but is not as versatile as the use of Gdxxrw as described below.
- Xlexport requires that the set elements have already been specified in the spreadsheet and must be matched up.
- Xlexport sends whatever data is current for an item at the stage of the program where the libinclude occurs and reflects any previous calculations and model solutions.

© 2022 Prof. Bruce McCarl

- The merge option leaves prior data alone but does wipe out any formulas in the Xlexport range.
- The range should be explicitly specified if one intends to not have the procedure wipe out the data (if merge is off) and formulae from one row below and one column to the left of the insertion point (if just j1 is specified then the spreadsheet is wiped from k2 to the bottom and left.

16.6.1.3 Xldump

The procedure Xldump can be used export data from a GAMS program to a spreadsheet. It writes data and labels to a specified range overwriting what ever is there. GAMS internal rules controls the order of the items in the output as discussed in the chapter on Rules for Item Capitalization and Ordering.

Example:

(fromExcel.gms)

```
Solve tranport using lp minimizing totalcost ;
$libinclude xldump transport.l myspread.xls output2!a1
$libinclude xldump transport.l myspread.xls output!a1..d4
$libinclude xldump transport.l myspread.xls output4!a1
```

This copies the data for the solution levels (.l) of the variable named transport into the spreadsheet called **myspread.xls** in the ranges specified. The first command exports the data after the solve statement into the sheet called output2 in the range starting with a1 and clears the rest of that sheet. The second exports the data after the solve statement into the sheet called output in the range a1 to d4 leaving the rest of the sheet alone. The third exports the data after the solve statement into the sheet called output4 and will create that sheet if it does not already exist.

The output2 sheet before the import looks like

myspread.XLS

	A	B	C	D	E	F	G
1	I am going to be gone					I will be erased	
2							
3							
4							
5							
6							
7							
8	I will be erased						
9							

and the output2 sheet afterward looks like

© 2022 Prof. Bruce McCarl

myspread.XLS

	A	B	C	D	E	F	G
1		newyork	chicago	topeka			
2	seattle	275	75				
3	sandiego	50		250			
4							
5							
6							
7							
8							
9							

while the sheet named output looks like

myspread.XLS

	A	B	C	D	E	F	G
1		newyork	chicago	topeka		I will not be erased	
2	seattle	275	75				
3	sandiego	50		250			
4							
5							
6							
7							
8	I will not be erased						
9							

Also the sheet output4 is created, as it did not previously exist.

Notes:

The order of the results is controlled by internal rules in GAMS as discussed in the chapter on Rules for Item Capitalization and Ordering.

- When the range is specified just as the upper left corner any entries below and right will be cleared. You must use a fully specified range if you wish to avoid this.
- The use of Xldump either requires the spreadsheet to be closed or it to be shared through the Excel Tools Share Workbook option. However, if shared, the data put by GAMS will only be reflected in the workbook if you do a file save and the procedure may be slow. Often the best option is to close the workbook.
- The use of Xldump permits simplicity of data transfer but is not as versatile as the use of Gdxxrw as described below.
- Xldump places the whole data parameter into the spreadsheet.
- Xldump sends whatever data is current for an item at the stage of the program where the libinclude occurs and reflects any previous calculations and model solutions.

16.6.2 Gdxxrw

Gdxxrw is a GAMS Corporation developed utility to read and write Excel spreadsheet data using GDX files. In turn the associated GDX files can be read or written by GAMS as discussed in the Using

© 2022 Prof. Bruce McCarl

GAMS Data Exchange or GDX Files chapter.

Gdxxrw is designed to be called from a DOS prompt. It can also be invoked using the GAMS $Call or Execute commands as discussed above. There are a lot of options involved with Gdxxrw that are described in the document *GDXXRW*. Here we just give a broad treatment of the most important.

GDXXRW can work with .xls, .xlsb, .xlsx and .xlsm formats, GDXXRW will write an Excel files as a .xlsx file unless a different file extension is specified for the output file.

Gdxxrw uses command line arguments. The basic calling sequence is

```
Gdxxrw Inputfile Output=filename options
```

where

Inputfile	must have an extension and tells the name of a file to read from or write to. The read from a spreadsheet occurs if the extension is a workbook extension (.xls, .xlsb, .xlsx, .xlsm, .wk1, .wk2, .wk3 and .dbf). The write to the spreadsheet occurs if the file has a GDX extension. This can also be entered as I=inputfile.
Output=filename	is an optional specification of the name of target workbook or GDX file where the output is to be written. If not present the file name will be the input file name with a workbook (.XLS or .XLSX, .XLSB, XLSM) or GDX extension and no path. A shortcut entry occurs with O=filename.
Options	are a number of possible options as discussed below.

Notes:

- If no path is specified for the input or output files the file is assumed in the current or project file directory.
- One needs to use quotes for file names and paths with spaces in them as discussed above.
- Command line parameters can be read from a file or a spreadsheet as discussed below.

16.6.2.1 Command line parameters

The applicable command line parameters in part depend on whether one is reading from or writing to a spreadsheet. Thus, I treat these items separately for each case. First, however I treat parameters common to both reading and writing involving

- The target spreadsheet range.
- How one handles GAMS items of various dimensions.
- How one identifies the names of and types of items to transfer.

16.6.2.1.1 Rng=

To write data to or read data from a spreadsheet a range needs to be specified for each GAMS item to

© 2022 Prof. Bruce McCarl

be transferred. A range is specified using the syntax:

```
Rng=SheetName!CellRange
```

where

SheetName!CellRange is standard Excel notation

Rng= (where capitalization does not matter) alerts Gdxxrw that a range name is following

Sheetname is the name of a sheet within the Excel workbook

! is the exclamation point symbol

CellRange is specified by using TopLeft:BottomRight or TopLeft..BottomRight or Rangename cell notation like A1:C12 or B1..Q221 or Myrange.

but a complete specification is not required. In particular, when

- Sheetname and CellRange or the whole Rng= entry are omitted the first sheet cell A1 is addressed
- Sheetname is omitted the first sheet is addressed i.e. use of Rng=B2 would address the first sheet in the B2 cell (Note this can be altered using the NameConv or NC parameter)
- Sheetname is specified and the sheet does not exist, then
  - when writing to a spreadsheet a new sheet will be added to the workbook with that name.
  - when reading from a spreadsheet an error will occur if the named sheet is not present.
- CellRange is omitted cell A1 is addressed
- !CellRange appears the range identified by Cellrange is addressed in the first sheet
- CellRange is specified by Rng=TopLeft only then the program will extend the range as far down and to the right as needed wiping out all sheet contents below and to the right of the starting point. (this is also controlled by the skipempty parameter as discussed below)
- CellRange can be specified as Rng=namedrange and the program will search for a predefined Excel range with that name. When the name does not exist an error will occur.

16.6.2.1.2 NameConv=: NC=

The interpretation of ranges that do not contain an "!" can be changed through use of the naming convention command line parameter. This is specified as follows

```
NameConv=integer
```

or

```
NC=integer
```

Valid values for integer are

© 2022 Prof. Bruce McCarl

0 to interpret symbols without ! as CellRanges
1 to interpret symbols without ! as Sheetnames

and the default value is zero.

16.6.2.1.3 GAMS item dimension: Dim=, Rdim=, Cdim=

Nominally, the Gdxxrw program is set up to read and write a 2 dimensional data item i.e. a(i,j). When the object is more or less than 2 dimensional, then the user must specify the dimension at hand and the way the data are arrayed in the spreadsheet. Three command line parameters control this

- Dim tells the total dimension i.e. 1 for a vector like r(i), 2 for a two dimensional matrix like a (i,j), 3 for a three dimensional item like b(i,j,k).
- Cdim dimensions to be placed in columns of spreadsheet where the first Cdim rows of the data range will be used for labels.
- Rdim dimensions to be placed in rows of spreadsheet where the first Rdim columns of the data range will be used for labels.

These are specified as

```
Dim=integer
Rdim=integer
Cdim=integer
```

Note they do not all need to be specified Dim equals the sum of Cdim and Rdim. and if both Cdim and Rdim are omitted, the program assumes that Cdim = 1 and Rdim= Dim – 1.

The set up of the Rdim and Cdim parameters is much like the option statement associated with a display as discussed in the Improving Output via Report Writing chapter. Their use is illustrated below.

16.6.2.1.4 Data specification

Data items can be read from the spreadsheet in the form of set elements, set explanatory text, parameters and data for variables or equations. Such entries have both general and specific requirements. In general the commands are of the form

```
DataType=ItemName Rng=DataRange Dimensions SymbolOptions
```

where

DataType	identifies the type of item and is one of the symbols: Par, Set, Dset, Equ, Text, or Var as discussed below.
Itemname	identifies the name of the item in the GDX file. These should follow the GAMS item naming rules as discussed in the Rules for Item Names, Element Names and Explanatory Text chapter.
Rng=Datarange	is an optional specification of the range where the item is to be placed or withdrawn from following the practices above where if omitted is cell A1 of the first sheet in the workbook.

© 2022 Prof. Bruce McCarl

Dimensions	is an optional specification of item dimensions following the dimension identification practices identified in the section just above.
SymbolOptions	are optional and controls read or write specific options.

There are multiple forms of this command and each will be discussed separately.

16.6.2.1.4.1 Writing Text and Links

GDXXRW allows one to write text or hyperlinks into a spreadsheet

This is done using the syntax

Text = "String of characters" Range=place

or

Text = "String of characters" Range=place Link=Address

or

Text = "String of characters" Range=place LinkID=gamsitem

where the text encased between the " marks is written to the specified range
if link=address is specified then the text is made a hyperlink referring to the location address
if linkid=gamsitem is specified then the link points to the upper left hand corner of the place where that item is located in a GDXXRW instruction that is in same GDXXRW execution

Example:

(gdxxrwwrite.gms)

```
execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls text="This is the link dictionary

execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls text="This is the twodim paramete
execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls par=twodim rng=output!b2 text="Li

execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls text="This is the threedim parame
execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls par=threedim rng=output!a12 text=

execute 'gdxxrw gdxxrwwrite.gdx o=gdxxrwss.xls text="Link to threedim second var
```

This labels items in the spreadsheet**gdxxrwss.xls** some with pure text and insetrts three hyperlinks.on a sheet called dictionary.

Notes:

- The LinkID must be in the same command as the SET, PAR etc that puts the gamsitem
- The text must be encased between " marks.

© 2022 Prof. Bruce McCarl

16.6.2.1.4.2 Set data: Set= and Dset=

Sets may be input in two ways depending on whether to permit accommodation of duplicate entries. In output DSET and SET are the same. The syntax is

```
Set =nameofset  Rng=DataRange Dimensions Values=valueoptions SymbolOptions
Dset=nameofset  Rng=DataRange Dimensions SymbolOptions
```

where

Set= identifies one is to input or output set elements and optionally an associated set of element explanatory text or indicators. on input duplicates will cause read errors. When writing the element name is always written and the explanatory text will be written if the range specification permits. The keyword Set also is associated with option values which indicates how non zero entries in the range are to be interpreted.

Namely when data are **input** and valueoptions equals

Auto Based on the range, row and column dimensions for the set, the program decides on the value type to be used selecting from dense or YN . This is the default for Values. When Auto or a values entry is not entered and thus the default is active then the import of data behaves differently depending on the vaules of rdim and cdim. Namely If

Only one of Rdim and Cdim are non zero (ie if Rdim = 0 or Cdim = 0 and the data are in vector form) then the import behaves as if Values=Dense

Both Rdim and Cdim are non zero and the data are in tabular form then the import behaves as if Values=YN

NoData The data entries in the range for the set will be ignored and all entries will be included without reading explanatory text.

YN Only those items will be included that have a non empty data cell in the range where the cell contents that do not equal '0', 'N' or 'No'.

Sparse Only those items will be included that have a non empty data cell in the range. The string in the data cell will be used as the explanatory text for the set entry.

Dense All items will be included that are in the range regardless of whether they have entries . Any strings in the data cells in the range will be used as the explanatory text for the set elements.

String Obsolete: Now the same as Dense.

All Obsolete: Now the same as Dense.

However when data are **output** and the valueoptions equals

Auto Based on the range, row and column dimensions for the set, the program decides on the value type to be used selecting from dense or YN . This is the default for Values.
When Auto or a values entry is not entered and thus the default is active then the import of data behaves differently depending on the vaules of rdim and cdim. Namely If

Only one of Rdim and Cdim are non zero (ie if Rdim = 0 or Cdim = 0 and the data are in vector form) then the import behaves as if

© 2022 Prof. Bruce McCarl

Values=Dense
Both Rdim and Cdim are non zero and the data are in tabular form then the import behaves as if Values=YN

NoData	The cells in the range for the set will be blank.
YN	Only those items will be included that have a non empty data and the cell value in the range is Y if the element exists.
Sparse	Only those items will be included that have a non empty data and the cell value in the range is the explanatory text if the element exists.
Dense	Only those items will be included that have a non empty data and the cell value in the range is the explanatory text if the element exists.
String	Obsolete: Now the same as Dense.
All	Obsolete: Now the same as Dense.

Dset= Reads a set of strings from a field in the spreadsheet and enters the unique ones into the set. Duplicate labels in the range specified do not generate an error message. Dset cannot be used to write to the spreadsheet. The values parameter is not used for Dset.

Examples of data input:

(gdxtuple.gms and trytuple.xlsx)

In the input sheet of the **trytuple.xlsx** file we have the data for the sets to be imported

	A	B	C	D	E
1	test data for tuple input				
2		1	2	3	4
3	a	hello		no	yes
4	b		w2	y	
5	c	n		z2	
6	d	y			z
7					
8			11	12	13
9	yc	ga	aa		
10	yc	gb		ww	22
11	yd	gb		ab	ac
12					
13					
14		bd	be	bd	be
15		21	21	22	23
16	aa	aa			
17	ab		ab	ac	

© 2022 Prof. Bruce McCarl

and we have an index portion of the sheet (as discussed here) telling us how to import with comments at the far right

	I	J	K	L	M	N	O	P	Q	R	S	T	U
2				rdim	cdim	values		Comment					
3	set	aa1	input!a3:a6	1	0			reads set for rows of first tuple under default					
4	set	aa1v2	input!a3:a6	1	0	yn		reads set for rows of first tuple under yn					
5	set	aa1v3	input!a3:a6	1	0	dense		reads set for rows of first tuple under dense					
6	set	aa1v4	input!a3:a6	1	0	sparse		reads set for rows of first tuple under sparse					
7	set	aa1v5	input!a3:a6	1	0	nodata		reads set for rows of first tuple under nodata					
8	set	aa1v6	input!a3:a6	1	0	auto		reads set for rows of first tuple under auto					
9	set	ab1	input!b2:e2	0	1			reads set for columns of first tuple under default					
10	set	tuple1	input!a2:e6	1	1			reads first tuple under default					
11	set	tuple1v2	input!a2:e6	1	1	yn		reads first tuple under yn					
12	set	tuple1v3	input!a2:e6	1	1	dense		reads first tuple under dense					
13	set	tuple1v4	input!a2:e6	1	1	sparse		reads first tuple under sparse					
14	set	tuple1v5	input!a2:e6	1	1	nodata		reads first tuple under nodata					
15	set	tuple1v6	input!a2:e6	1	1	auto		reads first tuple under auto					
16	dset	b1	input!a9:a11	1	0			reads sets for second three dimensional tuple first row dimension					
17	dset	b2	input!b9:b11	1	0			reads sets for second three dimensional tuple second row dimension					
18	dset	b3	input!c8:e8	0	1			reads sets for second three dimensional tuple column dimension					
19	set	tuple2	input!a8:e11	2	1			reads second three dimensional tuple under default					
20	set	tuple2v2	input!a8:e11	2	1	dense		reads second three dimensional tuple under dense					
21	set	tuple2v3	input!a8:e11	2	1	sparse		reads second three dimensional tuple under sparse					
22	dset	c1	input!a16:a17	1	0			reads sets for third three dimensional tuple row dimension					
23	dset	c2	input!b14:e14	0	1			reads sets for third three dimensional tuple first column dimension					
24	dset	c3	input!b15:e15	0	1			reads sets for third three dimensional tuple second column dimension					
25	set	tuple3	input!a14:e17	1	2			reads third three dimensional tuple under default					
26	set	tuple3v2	input!a14:e17	1	2	dense		reads third three dimensional tuple under dense					
27	set	tuple3v3	input!a14:e17	1	2	sparse		reads third three dimensional tuple under sparse					

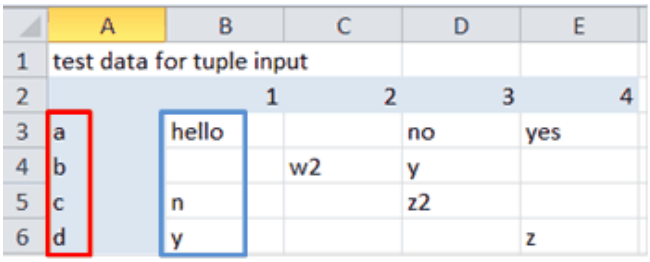
then we import the sets in the gams program **gdxtuple.gms** using the commands

```
$call gdxxrw trytuple.xlsx o=gdxtuple.gdx index=input!i2
$gdxin gdxtuple.gdx
$load aa1,aa1v2,aa1v3,aa1v4,aa1v5,aa1v6,ab1,tuple1
$load          tuple1v2
$load          tuple1v3
$load          tuple1v4
$load          tuple1v5
$load          tuple1v6
$load          b1
$load          b2
$load          b3
$load          tuple2
$load          tuple2v2
$load          tuple2v3
$load          c1
$load          c2
$load          c3
$load          tuple3
$load          tuple3v2
$load          tuple3v3
$gdxin
```

Now let us examine some of the consequences of the value commands.

First let us look at the alternative results for importing the set in the red box below and also look at the effect of the entries in the blue box

© 2022 Prof. Bruce McCarl



	A	B	C	D	E
1	test data for tuple input				
2		1	2	3	4
3	a	hello		no	yes
4	b		w2	y	
5	c	n		z2	
6	d	y			z

When read with the values entry being blank (or the default - auto condition) the put file at the bottom of the gms file for the set aa1 shows it to contain

```
Defined Elements      Explanatory Text
              a                hello
              b                b
              c                n
              d                y
```

which shows the reading process went beyond the specified range to the adjacent column and took entries there as explanatory text using blank entries when none was entered. Note when outputting to a spreadsheet or using .te in writing a put file that the element name (.tl) is used when the entry is blank. Also using auto and dense yields the same results.

When read with the values entry being yn we get

```
Defined Elements      Explanatory Text
              a                a
              d                d
```

Note here only a and d entries are defined as they have text entries in the blue column that and not N. Also this shows that the yn entry is a misnomer but rather the program accepts any entry which is not blank or N or NO or 0.

When read with the values entry being sparse we get

```
Defined Elements      Explanatory Text
              a                a
              d                d
```

Note here only a and d entries are defined as they have text entries in the blue column that and not N. Also this shows that the yn entry is a misnomer but rather the program accepts any entry which is not blank or N or NO or 0.

When read with the values entry being nodata we get

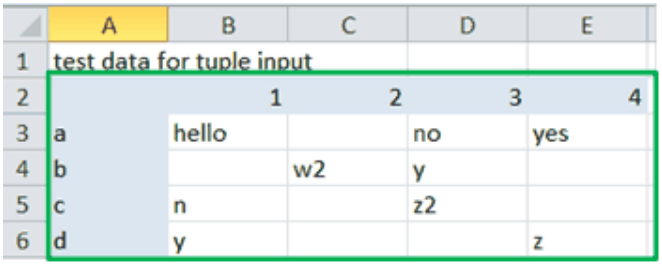
```
Defined Elements      Explanatory Text
              a                a
              b                b
              c                c
              d                d
```

Here all elements are defined as under dense but the explanatory text is not read and only element names are used.

Now suppose we turn to two dimensional sets or more generally tuples.

When reading tuple1 from the spreadsheet area in the green box below

© 2022 Prof. Bruce McCarl



	A	B	C	D	E
1	test data for tuple input				
2		1	2	3	4
3	a	hello		no	yes
4	b		w2	y	
5	c	n		z2	
6	d	y			z

If we read it with the values entry being blank (or the default - auto condition) the put file at the bottom of the gms file for the set tuple1 shows it to contain

Defined Elements	Explanatory Text
a.1	a.1
a.4	a.4
b.2	b.2
b.3	b.3
c.3	c.3
d.1	d.1
d.4	d.4

which shows the reading process operated as yn only defining entries with non blank text and not using that text for explanatory text. Also using auto and yn yields the same results.

When read with the values entry being dense we get

Defined Elements	Explanatory Text
a.1	hello
a.2	a.2
a.3	no
a.4	yes
b.1	b.1
b.2	w2
b.3	y
b.4	b.4
c.1	n
c.2	c.2
c.3	z2
c.4	c.4
d.1	y
d.2	d.2
d.3	d.3
d.4	z

Note here only all possible entries are defined with the non empty elements being used as explanatory text and when that explanatory text is blank then the element tuple becomes is used for the explanatory text when it is written though put or other means .

When read with the values entry being sparse we get

Defined Elements	Explanatory Text
a.1	hello
a.3	no
a.4	yes

© 2022 Prof. Bruce McCarl

b.2	w2
b.3	y
c.1	n
c.3	z2
d.1	y
d.4	z

which defines tuple entries when there is associated text entries including those with N or no.

When read with the values entry being nodata we get

Defined Elements	Explanatory Text
a.1	a.1
a.2	a.2
a.3	a.3
a.4	a.4
b.1	b.1
b.2	b.2
b.3	b.3
b.4	b.4
c.1	c.1
c.2	c.2
c.3	c.3
c.4	c.4
d.1	d.1
d.2	d.2
d.3	d.3
d.4	d.4

Here all elements are defined as under dense but the explanatory text is not read.

Examples of data output:

This was also run to output items **(gdxtuple.gms and trytuple.xlsx)** with GDXXRW taking instructions from the index in the output sheet of**trytuple.xlsx** .

When putting out the set tuple1v4 with the value being blank or set to yn or set to auto we get

	A	B	C	D	E
35		1	2	3	4
36	a	Y		Y	Y
37	b		Y	Y	
38	c	Y		Y	
39	d	Y			Y

When putting out the set tuple1v4 with the value set to sparse or dense or string or all we get

© 2022 Prof. Bruce McCarl

55		1	2	3	4
56	a	hello		no	yes
57	b		w2	y	
58	c	n		z2	
59	d	y			z

When putting out the set tuple1v4 with the value set to nodata we get where the range entries are blank.

75		1	2	3	4
76	a				
77	b				
78	c				
79	d				

Comments:

- The explanatory text is ignored under the nodata option
- When inputting sets only elements with associated text are defined under yn and sparse
- When inputting sets the meanings of the commands have changed over time with the current entries not being. In particular
  - Before version 24.3 the command strings was the same as dense above
  - Starting with 24.3 the command strings became the same as sparse above
  - Starting with 24.4.1 the command all was introduced and it functioned like dense above
  - Starting with 24.4.6 the command structure for values became what is documented above with the option string redefined
  - With these changes GAMS broke backward compatibility. Because of this, users who employed the values option for the versions before 24.4.6. need to review the ways their input is working and in the case where it is not delivering what is desired then either change the parameters for the GDXXRW call to the appropriate one of those described above or change the workbook data .

A number of examples are appropriate here.

Loading rows of set elements

© 2022 Prof. Bruce McCarl

Loading columns of set elements
Loading set elements only if they have data or text
Writing set elements
Sets and explanatory text – use of Set
Loading by upper left hand corner
Loading sets from data tables
Loading sets from lists with duplicates
Dealing with a tuple
Execution time set reads
Execution time set writes
Loading the set into GAMS
Unloading the set from GAMS

Suppose I have a list of set elements that I wish to read in a spreadsheet such as the section below from **gdxxrwss.xls**.

	A	B	C	D	E
1	set I1 to read as row with set, i1a with dset				
2	trains	cars	planes		
3					

Either of the following 2 statements read it and place it into a GDX file at compile time as in **gdxxrwread.gms**

```
$call "Gdxxrw gdxxrwss.xls set=i1 Rng=a2:c2 cdim=1"
$call "Gdxxrw gdxxrwss.xls dset=i1a Rng=a2:c2 cdim=1"
```

On the other hand if the elements were listed in a column as in sheet1 of **gdxxrwss.xls**.

	A	B
34	j1 with set	
35	truck	
36	ship	
37	rail	
38		

One can read this with either of the below (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls set=j1 Rng=sheet1!a35:a37 rdim=1"
$call "Gdxxrw gdxxrwss.xls dset=j1a Rng=sheet1!a35:a37 rdim=1"
```

Suppose a spreadsheet contains a potential list of elements names and one only wishes those element names associated with nonzero data or yes such as the section below from **gdxxrwss.xls**.

	A	B	C	D	E
5	boats	watercraft	itwillskipm	what	
6	yes	1		no	
7					

© 2022 Prof. Bruce McCarl

One would read this with the statement **gdxxrwread.gms**

```
$call "Gdxxrw gdxxrwss.xls set=i2 rng=sheet1!a5:d6 cdim=1 values=yn"
```

and the resultant i2 set will only have the elements boats and watercraft since the other two elements are associated with a blank or a no.

One can save set elements to a spreadsheet called **gdxxrwss.xls** in rows using the commands (**gdxxrwwrite.gms**)

```
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=i Rng=output2!a1  cdim=1 "
```

where use of SET generates an associated row of Y entries with explanatory text. DSET cannot be used.

Similarly one can save the set elements to a spreadsheet in columns using the commands (**gdxxrwwrite.gms**)

```
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=j Rng=output2!a5  rdim=1"
```

where use of SET ordinarily generates an associated column or row with explanatory text unless the values parameter is defined

```
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=i rng=output2!a16:d16  cdim
```

where the element and an yes no element indicator will occur.

Finally if the range only leaves room for element names then that is all that will be output as in the statement below.

```
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=i Rng=output2!a15:c15  cdim
```

DSET cannot be used.

Explanatory text is read using the command (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls set=i3 Rng=sheet1!a9:e10 cdim=1"
```

and a spreadsheet segment like the following from sheet1 of **gdxxrwss.xls**

	A	B	C	D	E	F
9	new york	chicago	boston	skipme	skipme2	
10	city1	city2	city3	No		
11						

© 2022 Prof. Bruce McCarl

where the resulting set and its explanatory text elements are

```
Element     Explanatory text

new york     city1
chicago      city2
boston       city3
skipme       No
skipme2      skipme2
```

Note here the skipme2 explanatory text is just the element name as it has a blank entry for the explanatory text.

Sets can be addressed just indicating the upper left hand corner as follows (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls se=0 dset=i4 Rng=sheet1!a13 cdim=1"
$call "Gdxxrw gdxxrwss.xls se=0 set=i4a Rng=sheet1!a13 cdim=1"
$call "Gdxxrw gdxxrwss.xls se=0 set=i5 Rng=sheet1!a16 cdim=1"
```

This is best done with the skipempty=0 or se=0 option as otherwise Gdxxrw can read more of the sheet and cause errors.

One may wish to enter a data table and take the sets from it. Given a spreadsheet segment like the following from sheet1 of **gdxxrwss.xls**

	A	B	C	D	E
20		cleveland	chicago	dallas	
21	brussels	5000	6000	0	
22	san francis	2200	2000	1800	
23	boston	700	900	1500	
24					

one can take the set across the top with any of the following

```
$call "Gdxxrw gdxxrwss.xls set=i6 Rng=sheet1!b20:d20 cdim=1"
$call "Gdxxrw gdxxrwss.xls dset=i6a Rng=sheet1!b20:d20 cdim=1"
$call "Gdxxrw gdxxrwss.xls set=i6c Rng=sheet1!b20:d21 cdim=1"
```

One can also take a set vertically from Column A as follows (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls dset=j4 Rng=sheet1!a21:a23 rdim=1"
```

One may wish to extract set elements from a spreadsheet where there is no unique list of elements that can be read but rather a list where the name is repeated. In the example below note that in rows 26 and 27 there are set elements names but they are duplicated (**gdxxrwss.xls**)

© 2022 Prof. Bruce McCarl

	A	B	C	D	E	F
26		brussels	brussels	san francis	san francisco	
27		cleveland	chicago	cleveland	chicago	
28	ship	5000	6000	0	0	
29	truck	0	0	2200	2000	
30	rail	0	0	2200	2000	
31	barge	0	0	2800	2800	
32						

One can read this with DSET as follows (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls dset=i7 Rng=sheet1!b26:e26 cdim=1"
$call "Gdxxrw gdxxrwss.xls dset=i8 Rng=sheet1!b27:e27 cdim=1"
```

One may wish to read or write a tuple. They can be read using SET (**gdxxrwread.gms**) but DSET cannot be used.

```
$call "Gdxxrw gdxxrwss.xls set=i10 Rng=sheet1!b26:e27 cdim=2"
execute "Gdxxrw gdxxrwss.xls set=i10a Rng=sheet1!b26:e27 cdim=2"
```

and written as follows (**gdxxrwwrite.gms**)

```
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=ii Rng=output2!a10  rdim=2
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=ii Rng=output2!a20  cdim=2
```

where rdim causes the writing to be oriented in rows and cdim in columns. The written tuple would also be associated with an added row or column containing explanatory text.

All of the statements up to now have used $Call which executes the Gdxxrw command at compilation time but some users may wish to load sets at execution time. This is limited to subsets that are dynamic sets and cannot be used in domains. To do this one simply uses the statements as above, but substitutes Execute in place of $Call as follows (**gdxxrwread.gms**) where instead of

```
$Call "Gdxxrw gdxxrwss.xls set=i9 Rng=sheet1!b20:c21 cdim=1"
```

one uses

```
Execute "Gdxxrw gdxxrwss.xls set=i9 Rng=sheet1!b20:c21 cdim=1"
```

When loading sets it is often desirable to use $Call and allows domain definitions. This is hardly ever desirable when unloading or writing to a spreadsheet. In such case one uses the Execute syntax as follows (**gdxxrwwrite.gms**)

```
Execute "Gdxxrw gdxxrwss.xls set=i9 Rng=sheet1!b20:c21 cdim=1"
```

Getting a set from the spreadsheet into a GDX file is only half the battle. One must also use commands in GAMS to load the data as discussed in the chapter Using GAMS Data Exchange or GDX Files. At

© 2022 Prof. Bruce McCarl

compile time this is done using (**gdxxrwread.gms**)

```
set i1;
$call "Gdxxrw gdxxrwss.xls set=i1 Rng=sheet1!a2:c2 cdim=1"
$GDXin gdxxrwss.gdx
$load i1
```

where the set must be declared in a set statement then one can if needed create the GDX file using GDXRW, then one uses a Gdxin to identify the source and a Load to bring in the data.

At execution time one does the following (**gdxxrwread.gms**)

```
set i9(i6a);
*set from data
execute "Gdxxrw gdxxrwss.xls set=i9 Rng=sheet1!b20:c21 cdim=1"
Execute_load 'gdxxrwss' i9;
```

where the set must be declared as a subset in a set statement then one can if needed create the GDX file using execution time GDXRW, and an Execute_Load to bring in the data with an identification of the GDX source file name.

One can load the universe of labels from a GDX file into a set at run-time using the syntax

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

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

Getting a set to the spreadsheet from a GDX file is also only half the battle. One must also use commands in GAMS to place the data into the GDX file as discussed in the chapter Using GAMS Data Exchange or GDX Files. This should generally not be done at compile time so one should only use the execute command as follows following (**gdxxrwwrite.gms**)

```
Execute_unload ' ' threedim,i,j,k,ii;
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls set=i Rng=output2!a1  cdim=1 "
```

where the Execute_Unload tells what data to place in the GDX file and identifies the GDX source file name. The matching gdwxrw execution tells the name of the GDX file, the name of the spreadsheet and identifies the data to unload.

Notes:

- Either the row or the column dimension (Rdim or Cdim) should be set to '1' to specify a row or column for the set.
- One must be careful when using Gdxxrw as each time the command is executed the GDX file is erased and only has the current contents and thus should be written just before if reusing the name.

© 2022 Prof. Bruce McCarl

- One also obtains output of sets using the command Execute_Unloaddi where the GDX file automatically includes all sets associated with unloaded parameters, variables and equations without need to list the set element names (see the example **gdxexectrnsport.gms** and the resultant GDX file **tran2di.gdx**) where the statement `execute_unloaddi 'tran2di',d, x, item;` is used and the sets i,j,k appear in the GDX file even though they were not specified.

16.6.2.1.4.3 Parameter data: Par

When an item containing data along with identifying set elements is to be read or written one uses

```
Par=parametername Rng=DataRange Dimensions SymbolOptions
```

Examples:

Basic parameter writing is done using commands like (**gdxxrwwrite.gms**)

```
execute_unload 'Gdxxrwwrite.gdx' twodim,threedim,i,j,k,ii;
execute "Gdxxrw Gdxxrwwrite.gdx o=gdxxrwss.xls par=twodim Rng=output!a1"
```

which first creates the GDX file called GdxxrwWrite.gdx then uses the Gdxxrw program to save the parameter called TWODIM into the sheet called output starting at the upper left-hand corner A1 while drawing data from GdxxrwWrite.gdx and writing into the spreadsheet called GdxxrwSS.xls. The resultant spreadsheet segment in the output sheet is as follows

	A	B	C	D	E
1		j1	j2	j3	
2	i1	101	102	103	
3	i2	201	202	203	
4	i3	301	302	303	
5					

Parameters can be read the using essentially the same command with the name of the source spreadsheet appearing after Gdxxrw as follows (**gdxxrwread.gms**)

```
$call "Gdxxrw gdxxrwss.xls par=distance Rng=sheet1!a20:d23 rdim=1 cdim=1"
$GDXin gdxxrwss.gdx
$load distance
execute "Gdxxrw gdxxrwss.xls par=distance2 Rng=sheet1!a20:d23"
execute_load 'gdxxrwss.gdx' distance2
```

Notes:

- When writing to a GDX file and moving the data on to a spreadsheet, one should use the GAMS Execute_Unload and Execute commands to ensure that the current parameter data present at the time the Execute_Unload is saved in the GDX file.
- One needs to watch out for file names and paths with spaces in them as discussed above.
- Use of $Call would write out the data values at compile time not after calculations.

© 2022 Prof. Bruce McCarl

- One must use Execute for the Gdxxrw command whenever an Execute_Unload is used. Otherwise the program will use the last version of the GDX file written by a compile time $Call or a pre-existing GDX file in existence before the program ran. The GDX file created any Execute_Unload commands immediately before a $Call would not be used as all $ conditions are resolved at compile time then at the later execution time all Execute_Unload and Execute statements are resolved.
- GAMS chooses the row and column ordering as well as the way that table is arrayed in the spreadsheet. Users may contain additional control of the orders using the merge option as discussed below and additional control over the table layout using the CDIM and RDIM options as illustrated next.

By default Gdxxrw will choose a layout for multidimensional items that users may wish to change. Namely

- the last index position is included in a column and all other index positions are included in rows.
- Thus with a ten dimensional item, each spreadsheet row would depict one combination of the set elements within the first nine index positions and each column would represent an element of the last index position.

The layout may be altered using a combination of the CDIM and RDIM parameters. In particular suppose I have an item x(i,j,k) and wish to place it in or read it from a spreadsheet in varying layouts as follows (**gdxxrwwrite.gms**)

```
execute "Gdxxrw gdxxrwss.gdx par=threedim Rng=output!a1  cdim=3 rdim=0"
execute "Gdxxrw gdxxrwss.gdx par=threedim Rng=output!a8 cdim=2 rdim=1"
execute "Gdxxrw gdxxrwss.gdx par=threedim Rng=output!a15 cdim=1 rdim=2"
execute "Gdxxrw gdxxrwss.gdx par=threedim Rng=output!a30 cdim=0 rdim=3"
```

where the first creates a row vector of numbers with all indices varied in the columns as in the output page of the spreadsheet **gdxxrwss.xls**

	A	B	C	D	E	F	G	H
30	i1	i1	i1	i1	i1	i1	i1	i1
31	j1	j1	j1	j2	j2	j2	j3	j3
32	k1	k2	k3	k1	k2	k3	k1	k2
33	10101	10102	10103	10201	10202	10203	10301	10302
34								

the second a matrix with one index in the rows and two in the columns as follows

	A	B	C	D	E	F	G	H	I	J	K
38		j1	j1	j1	j2	j2	j2	j3	j3	j3	
39		k1	k2	k3	k1	k2	k3	k1	k2	k3	
40	i1	10101	10102	10103	10201	10202	10203	10301	10302	10303	
41	i2	20101	20102	20103	20201	20202	20203	20301	20302	20303	
42	i3	30101	30102	30103	30201	30202	30203	30301	30302	30303	
43											

and the last a column vector with many rows.

© 2022 Prof. Bruce McCarl

	A	B	C	D	E
60	i1	j1	k1	10101	
61	i1	j1	k2	10102	
62	i1	j1	k3	10103	
63	i1	j2	k1	10201	
64	i1	j2	k2	10202	
65	i1	j2	k3	10203	
66	i1	j3	k1	10301	
67	i1	j3	k2	10302	
68	i1	j3	k3	10303	
69	i2	j1	k1	20101	
70	i2	j1	k2	20102	
71	i2	j1	k3	20103	
72	i2	j2	k1	20201	
73	i2	j2	k2	20202	
74	i2	j2	k3	20203	
75	i2	j3	k1	20301	
76	i2	j3	k2	20302	
77	i2	j3	k3	20303	
78	i3	j1	k1	30101	
79	i3	j1	k2	30102	

Reading data from the spreadsheet entails essentially the same commands. For example, in **gdxxrwread.gms** the following lines appear which specify data layout

```
$call "Gdxxrw gdxxrwss.xls par=distance Rng=sheet1!a20:d23 rdim=1 cdim=1"
execute "Gdxxrw gdxxrwss.xls par=distance2 Rng=sheet1!a20:d23"
$call "Gdxxrw gdxxrwss.xls par=modedistance Rng=sheet1!a26:e31 rdim=1 cdim=2"
$call "Gdxxrw gdxxrwss.xls par=modedistance2 Rng=sheet1!a52:e56 rdim=2 cdim=1"
```

where the second line uses a default rdim=1 cdim=1. One may also use DIM as below

```
$call "Gdxxrw gdxxrwss.xls par=distance Rng=sheet1!a20:d23 dim=2 cdim=1"
execute "Gdxxrw gdxxrwss.xls par=distance2 Rng=sheet1!a20:d23 dim=2"
$call "Gdxxrw gdxxrwss.xls par=modedistance Rng=sheet1!a26:e31 rdim=1 dim=3"
$call "Gdxxrw gdxxrwss.xls par=modedistance2 Rng=sheet1!a52:e56 dim=3 cdim=1"
```

Notes:

- When writing to a spreadsheet, special values such as Eps, NA and Inf will be written but this can be changed as discussed below. When reading data from a spreadsheet, the ASCII strings for these special character stings will be used to write corresponding special values to the GDX file.
- Cells that are empty or zero will not be written to the GDX file.

The ordering of the set elements within the resultant spreadsheet or within GAMS are controlled by the GAMS element ordering rules and the unique element list as discussed in the Rules for Item Capitalization and Ordering chapter. But this may be controlled using merge as discussed below.

© 2022 Prof. Bruce McCarl

16.6.2.1.4.4 Variable and equation data: Equ and Var

When data associated with variables or equations are to be read or written one may use

```
Equ=equationname.attribute Rng=DataRange Dimensions SymbolOptions
Var=variablename.attribute Rng=DataRange Dimensions SymbolOptions
```

but these only work for going to the spreadsheet at the moment and one should pass such items through parameters.

Notes:

- When writing an attribute of a variable or equation can be addressed using the attribute names .L, .M, .Lo , .Up .Prior and .Scale
- The attribute names are not case sensitive.
- Today attributes of variables cannot be read from a spreadsheet and included in the GDX file. Rather one must bring the item into GAMS as a parameter and use a replacement statement. A revision is planned in the near future to allow this.
- Today one can only unload a whole variable into a GDX file then save selected attributes. A revision is planned in the near future to allow this.

16.6.2.1.5 Special options for reading from a spreadsheet: Skipempty= and Se=

Spreadsheet files may contain blank rows or columns as well as columns and rows you do not want to read.

For blank rows or columns the skipempty command controls the way blank rows or columns are handled and causes Gdxxrw to either stop when a blank row or column is encountered or skip over that. This is done using the syntax

```
SkipEmpty=integer
```

or

```
SE=integer
```

which must **precede any par, set etc statements that will be affected by it**

where

The integer value tells the number of maximum number of blank rows or columns that may be contained within a block of data and that integer value is one less that the number of blank rows or columns that if found will signal the end of the data block.

Allowable values are 0 to N, and the default is 1.

A value of 0 causes Gdxxrw to stop loading data when an empty row or column is encountered.

A value of 1 causes GDXXRW to skip over incidences of one blank row or column but terminate the read if 2 or more adjacent blank rows or columns are found.

© 2022 Prof. Bruce McCarl

When a the range is specified only in terms of the upper left corner (a single cell), and skip empty is set to zero an empty row or column will always indicate the end of the row or column range.

For rows or columns that you do not want to read the IgnoreColumns and IgnoreRows commands identify rows and columns to skip over. This is done using the syntax

```
IgnoreColumns=comma delimited list of spreadsheet column names
IgnoreRows=comma delimited list of spreadsheet row numbers
```

where

The comma delimited list of spreadsheet column names gives the names of columns in the range to skip over and is of the form A,C,Z,ZZ which would cause the reading to skip the columns labeled A,C,Z, and ZZ in the Excel worksheet

The comma delimited list of spreadsheet row names gives the names of rows in the range to skip over and is of the form 2,12,210 which would cause the reading to skip the rows labeled 2, 12 and 210 in the Excel worksheet

The specification of ignored rows or columns follows the symbol specification and only applies to that symbol.

Examples:

For skipempty suppose I have a spreadsheet table that has blank entries in it like the skipempty sheet of the workbook **gdxxrwss.xls**

	A	B	C	D	E	F	G
1							
2			ship	truck		rail	
3	brussels	cleveland	5000	0		0	
4	brussels	chicago	6000	0		0	
5	san francis	cleveland	0	2200		2200	
6							
7	san francis	chicago	0	2000		2000	
8							

and I read it using the default setting (**gdxxrwskipempty.gms**)

```
$call "Gdxxrw gdxxrwss.xls o=GDXse.gdx par=moded4 Rng=skipempty!a2:g69"
```

or

```
$call "Gdxxrw gdxxrwss.xls se=1 o=GDXse.gdx par=moded4 Rng=skipempty!a2:g69"
```

then after loading into GAMS the data become

```
                                ship       truck        rail

brussels      .cleveland     5000.000
brussels      .chicago       6000.000
san francisco.cleveland                   2200.000     2200.000
san francisco.chicago                     2000.000     2000.000
```

© 2022 Prof. Bruce McCarl

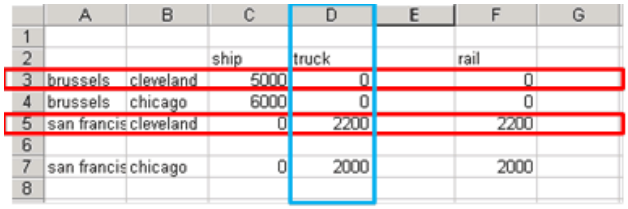
On the other hand if I read it with skipempty set to zero

```
$call "Gdxxrw gdxxrwss.xls se=0 o=GDXse.gdx par=moded3 Rng=skipempty!a2:g69 rd
```

the blanks terminate the read not reading the rail column and the san francisco.chicago row and the result is

```
                                    ship       truck

brussels      .cleveland        5000.000
brussels      .chicago          6000.000
san francisco.cleveland                    2200.000
```

For IgnoreRows and IgnoreColumns suppose we have a spreadsheet table that has rows and columns we do not want to read. For this we again employ the skipempty sheet of the workbook **gdxxrwss.xls** as in the screenshot below. Suppose in reading this we want to skip the rows in the red boxes and the column in the blue box



	A	B	C	D	E	F	G
1							
2			ship	truck		rail	
3	brussels	cleveland	5000	0		0	
4	brussels	chicago	6000	0		0	
5	san francis	cleveland	0	2200		2200	
6							
7	san francis	chicago	0	2000		2000	
8							

and we read it using the command (**gdxxrwskipempty.gms**)

```
$call "gdxxrw gdxxrwss.xls o=gdxignore.gdx se=0 par=moded5 rng=skipempty!a2:g69 rdim=2 cdim=1 IgnoreRows=3,5
```

then after loading into GAMS the data become

```
                                    ship        rail
brussels      .chicago          6000.000
san francisco.chicago                      2000.000
```

which omits entries from those rows and columns.

Note:

- The skipempty parameters must appear before any parameter, set, dset etc statements that use it and will persist for the rest of the statements in a command unless it is set to another value.
- The IgnoreRows and IgnoreColumns parameters appear after any parameter, set, dset etc GDXXRW command instruction and only affect reading of that item.

16.6.2.1.6 Special options for writing to a spreadsheet

When the input file has an extension '.gdx' then Gdxxrw will read data from a GDX file and in turn will write data to a '.xls' file interacting with Excel. In such the merge and clear command line parameters along with consideration of workbook open status are important.

© 2022 Prof. Bruce McCarl

16.6.2.1.6.1 Is the workbook open or shared?

When one wishes to write to a workbook a question arises as to whether that workbook can be open and how the workbook will look afterward if it was. Several comments are in order.

- A workbook cannot in general be open unless you have made special provisions with an error signaled indicating a file sharing conflict will arise when the target file is open in Excel.
- To avoid this the sharing conflict error the user must either close the file or indicate that the spreadsheet is a shared Excel workbook in using the Excel Tools Share Workbook dialogue.
- In an open shared workbook the contents are not updated until you have done a file save in Excel.
- Writing to a shared workbook can be painfully slow.
- In general it is best to close the workbook.

16.6.2.1.6.2 Merge

When writing to a spreadsheet one can control data handling and set matching using the Merge command line parameter. When 'Merge' is active the only data that will be written to the spreadsheet are those data for which the set element names match row and column labels that are in the spreadsheet already. Also under Merge spreadsheet cells for which there is no matching row/column pair will not be changed. Also element ordering is explicitly specified overriding the default that would occur as controlled by the GAMS element ordering rules and the unique element list as discussed in the Rules for Item Capitalization and Ordering chapter.

Example:

Suppose I have a range in a spreadsheet that appears as in the sheet2 page of the workbook **gdxxrwss.xls** with exactly the same contents in the range i1:n4.

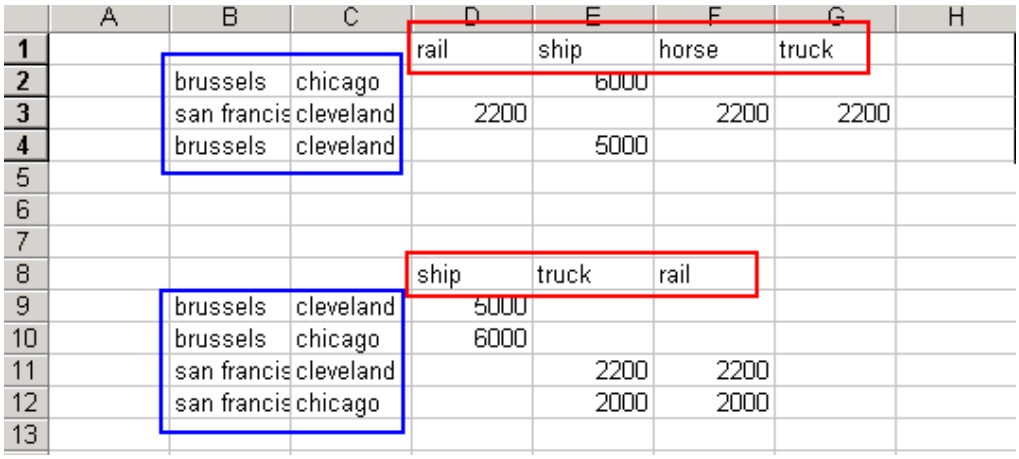
	A	B	C	D	E	F	G	H
1				rail	ship	horse	truck	
2		brussels	chicago					
3		san francis	cleveland			2200		
4		brussels	cleveland					
5								

and use the commands (**gdxxrwskipempty.gms**)

```
$call"Gdxxrw GDXse.gdx o=gdxxrwss.xls  par=moded4 Rng=sheet2!b1:g4 rdim=2 cdim
$call "Gdxxrw GDXse.gdx o=gdxxrwss.xls  par=moded4 Rng=sheet2!b8 rdim=2 cdim=1
```

then the resultant spreadsheet looks like

© 2022 Prof. Bruce McCarl



	A	B	C	D	E	F	G	H
1				rail	ship	horse	truck	
2		brussels	chicago		6000			
3		san francis	cleveland	2200		2200	2200	
4		brussels	cleveland		5000			
5								
6								
7								
8				ship	truck	rail		
9		brussels	cleveland	5000				
10		brussels	chicago	6000				
11		san francis	cleveland		2200	2200		
12		san francis	chicago		2000	2000		
13								

where the portion in rows b8-f12 is what happens without the merge and the part in rows 1-4 is what happened with it. Note that the column and row orders vary and the san francisco chicago row is missing since it is not mentioned in the labels before the merge operation and the horse column is present with it's data left alone.

Notes:

- Using the merge option will force the data to be presented in the order in which the row and column labels are entered.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a merge option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements.
- The matching of labels is not case sensitive.
- **Warning**: The Merge option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row / column headings in the GDX file. Cells containing strings or numbers are not affected.

16.6.2.1.6.3 Clear

When writing to a spreadsheet one can also use the Clear option to control data handling and set matching. When 'Clear' is active the only data that will be written to the spreadsheet are those data for which the set element names match row and column labels that are in the spreadsheet already but all data and formulas in the target range will be removed. Element ordering is explicitly specified overriding the default that would occur as controlled by the GAMS element ordering rules and the unique element list as discussed in the Rules for Item Capitalization and Ordering chapter.

Example:

Suppose I have a range in a spreadsheet just like that in the spreadsheet above for the merge example gdxxrwss.xls in the range i1:n4.

© 2022 Prof. Bruce McCarl

	I	J	K	L	M	N
1			rail	ship	horse	truck
2	brussels	chicago				
3	san francis	cleveland			2200	
4	brussels	cleveland				
5						

and use the commands (**gdxxrwskipempty.gms**)

```
$call"Gdxxrw GDXse.gdx o=gdxxrwss.xls  par=moded4 Rng=sheet2!i1 rdim=2 cdim=1
```

then the result is

	I	J	K	L	M	N	O
1			rail	ship	horse	truck	
2	brussels	chicago		6000			
3	san francis	cleveland	2200			2200	
4	brussels	cleveland		5000			
5							

which shows results similar to those under merge but the old data in the column labeled horse has been removed.

Notes:

- Using the clear option will force the data to be presented in the order in which the row and column labels are entered.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a clear option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements.
- The matching of labels is not case sensitive.
- **Warning**: The Clear option will clear all Excel formulas and values in the rectangle used, even if the cells do not have matching row / column headings in the GDX file.

16.6.2.1.6.4 Filter

You can cause GDXXRW to write data subject to an automatic filter specified in Excel.

By filtering in Excel, you can find values quicker and control not only what you want to see, but what you want to exclude. You can filter based on choices you make from a list, or you can create specific filters to focus on exactly the data you want to see. You can also search for text and numbers by using the **Search** box in the filter interface.

When you filter data in Excel, entire rows are hidden if values in one or more columns don't meet the filtering criteria. You can filter on numeric or text values.

© 2022 Prof. Bruce McCarl

The syntax for this involves adding

filter=n

where n=0 is the default condition and uses no filter

n=1 turns the filter and applies it to the row just above the data

n=2 turns the filter and applies it to the row 2 rows above the data

n=k turns the filter and applies it to the row k rows above the data

to the GDXXRW call.

An example is available in filter.gms

16.6.2.1.6.5 Special value and zero cell w riting options

A number of options can be used to alter the writing of special (INF, NA, EPS, UNDF) or zero values. **Each of these must appear before Par, Set etc statements that are affected by the settings.**

Epsout
Naout
Minfout
Pinfout
Undfout
Zeroout
Squeeze
Resetout

The parameter Epsout specifies the string to write to the spreadsheet when GAMS data to be written contains a value that would have been output as EPS (a number close to zero as set by the solver or subsequent calculations). The syntax is

```
EpsOut = String
```

where String gives the character string to use in place of values that GAMS would report as EPS. By default this is 'Eps'.

The parameter Naout specifies the string to write to the spreadsheet when data to be written contains a value that would have been output as NA (a number GAMS considers to be 'Not Available'). The syntax is

```
NaOut = String
```

© 2022 Prof. Bruce McCarl

By default this is 'NA'.

The parameter Minfout specifies the string to write to the spreadsheet when data to be written contains a value that would have been output as –INF (a value of negative infinity which generally is a lower bound on an unrestricted or non positive variable). The syntax is

```
MinfOut = String
```

By default this is '-Inf'.

The parameter Pinfout specifies the string to write to the spreadsheet when data to be written contains a value that would have been output as +INF (a value of positive infinity which generally is an upper bound on an unrestricted or non negative variable). The syntax is

```
PinfOut = String
```

By default this is '+Inf'.

The parameter Undfout specifies the string to write to the spreadsheet when data to be written contains a value that would have been output as Undf (a number GAMS could not calculate which it marked as 'Undefined'). The syntax is

```
UndfOut = String
```

By default this is 'Undf'.

The parameter ZEROOUT specifies the string to write to the spreadsheet when a variable or equation attribute is to be written that contains a value of zero. The syntax is

```
ZeroOut = String
```

By default this is '0'.

The SQUEEZE option tells Gdxxrw how to handle the writing of zero or default entries for attributes of variables and equations. A value for the field that is the default value for that variable or equation attribute will not be written to the spreadsheet. For example, the default for .L (Level value) is 0.0, and therefore zero will not be written to the spreadsheet. When I set SQ=0, all values will be written to the spreadsheet. The syntax is

```
Squeeze = integer
```

where Integer gives a zero or a one. The default value is one.

The RESETOUT option tells Gdxxrw to reset all the special value indicators. The syntax is

© 2022 Prof. Bruce McCarl

```
Resetout
```

16.6.2.2 Options for reading in command line parameters

The command line parameter input to Gdxxrw identifying where to put what can be quite long and impractical for inclusion on a command line. One also can stack multiple Gdxxrw actions into one job. Two options exist for this. One can place specify Gdxxrw options in a text file or a spreadsheet.

16.6.2.2.1 Command line parameters in a file

Command line parameters can be placed in a text file then read. This is indicated by including an entry in the program invocation that contains an @ followed by the name of the parameter file of instructions to read.

Example:

Suppose I wish to enter many of the Gdxxrw commands in the **gdxxrwread.gms** example into a file and execute all the commands with one execution of Gdxxrw. I can do this using a parameter file **gdxxrwparam.txt** and the GAMS code in the example **Gdxxrwread3.gms** as follows

```
$call 'gdxxrw gdxxrwss.xls  o=gdxall.gdx @gdxxrwparam.txt'
```

where the parameter file **gdxxrwparam.txt** is

```
se=0
set=i1          Rng=sheet1!a2:c2     cdim=1
dset=i1a        Rng=sheet1!a2:c2     cdim=1
set=i3          Rng=sheet1!a9:e10    cdim=1
dset=i4         Rng=sheet1!a13       cdim=1
se=0
set=i4a         Rng=sheet1!a13       cdim=1          dim=1
se=0
set=i5          Rng=sheet1!a16       cdim=1
set=i6          Rng=sheet1!b20:d20   cdim=1
dset=i6a        Rng=sheet1!b20:d20   cdim=1
dset=i6c        Rng=sheet1!b20:d21   cdim=1
dset=i7         Rng=sheet1!b26:e26   cdim=1
dset=i8         Rng=sheet1!b27:e27   cdim=1
set=i9          Rng=sheet1!b20:c21   cdim=1
set=i10         Rng=sheet1!b26:e27   cdim=2
set=i10a        Rng=sheet1!b26:e27   cdim=2
set=j1          Rng=sheet1!a35:a37   cdim=0   rdim=1
dset=j1a        Rng=sheet1!a35:a37   cdim=0   rdim=1
set=j2          Rng=sheet1!a40:b43   cdim=0   rdim=1
set=j3          Rng=sheet1!a46:b50   cdim=0   rdim=1
dset=j4         Rng=sheet1!a21:a23   cdim=0   rdim=1
dset=j5         Rng=sheet1!a53:a56   cdim=0   rdim=1
dset=j6         Rng=sheet1!b53:b56   cdim=0   rdim=1
par=distance    Rng=sheet1!a20:d23   cdim=1   rdim=1
par=distance2   Rng=sheet1!a20:d23
```

© 2022 Prof. Bruce McCarl

```
par=modedistance  Rng=sheet1!a26:e31    cdim=2    rdim=1
par=modedistance2 Rng=sheet1!a52:e56    cdim=1    rdim=2
par=modedistance3 Rng=skipempty!a2:g69  cdim=1    rdim=2
par=modedistance4 Rng=skipempty!a2:g69  cdim=1    rdim=2
```

that will control the Gdxxrw actions.

Notes:

- A parameter file can contain multiple lines to increase readability.
- When reading parameters from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.
- A put file can be used to write a parameter file can also be written during the execution of a GAMS model where one must use a Putclose as discussed in the Output using Put Commands chapter in the option file section or as implemented in Xlexport.gms and the subsequent Gdxxrw commands must use Execute so the put file is written before it is to be read (This will not happen with $Call).

16.6.2.2.2 Parameters in a spreadsheet

Command line parameters can be placed in a range of a spreadsheet then read. This is indicated by including the use of the Index command line parameter

```
Index = ExcelRange
```

Example:

Suppose I wish to integrate all the features of the **gdxxrwread.gms** example into an index area of a spreadsheet. I can do this using the myindex sheet of the **gdxxrwss.xls** spreadsheet and the GAMS code in the example **gdxxrwread2.gms** as follows

```
$call gdxxrw gdxxrwss.xls o=gdxall.gdx index=myindex!a1
```

where the myindex sheet looks like

© 2022 Prof. Bruce McCarl

	A	B	C	D	E	F	G	H
1				rdim	cdim	dim		
2	set	i1	sheet1!a2:c2		1			
3	dset	i1a	sheet1!a2:c2		1			
4	set	i3	sheet1!a9:e10		1			
5	dset	i4	sheet1!a13		1			
6							se=0	
7	set	i4a	sheet1!a13		1	1		
8	set	i5	sheet1!a16		1			
9							se=1	
10	set	i6	sheet1!b20:d20		1			
11	dset	i6a	sheet1!b20:d20		1			
12	dset	i6c	sheet1!b20:d21		1			
13	dset	i7	sheet1!b26:e26		1			
14	dset	i8	sheet1!b27:e27		1			
15	set	i9	sheet1!b20:c21		1			
16	set	i10	sheet1!b26:e27		2			
17	set	i10a	sheet1!b26:e27		2			
18	set	j1	sheet1!a35:a37	1	0			
19	dset	j1a	sheet1!a35:a37	1	0			
20	set	j2	sheet1!a40:b43	1	0			
21	set	j3	sheet1!a46:b50	1	0			
22	dset	j4	sheet1!a21:a23	1	0			
23	dset	j5	sheet1!a53:a56	1	0			
24	dset	j6	sheet1!b53:b56	1	0			
25	par	distance	sheet1!a20:d23	1	1			
26	par	modedistance	sheet1!a26:e31	1	2			
27	par	modedistance2	sheet1!a52:e56	2	1			
28							se=0	
29	par	modedistance3	skipempty!a2:g69	2	1			
30	par	modedistance4	skipempty!a2:g69	2	1			
31								

Notes:

- The parameters are read using the specified range, and treated as if they appeared directly on the command line.
- In the spreadsheet the first three columns of the range have a fixed interpretation: DataType (Par, Set, Dset, Equ, or Var), Item name identifier and spreadsheet data range. The fourth and following columns can be used for additional parameters like dim, rdim, cdim , merge, clear and skipempty. The column header contains the keyword when necessary, and the Cell content is used as the option value.
- When an entry appears in a column without a heading then it is directly copied into the Gdxxrw parameter file. Thus in the example the items in column G are directly copied into the file.
- Rows do not have to have entries in the first three columns if one just wants to enter persistent options such as skipempty or some of the special character string redefinitions.

© 2022 Prof. Bruce McCarl

Another Example:

The GAMS program **gdxxrwwrite2.gms** employs the use of an index area in a workbook in writing. Namely in the writeindex sheet of **gdxxrwss.xls** I have

	A	B	C	D	E	F	G
1							
2				rdim	cdim		
3	par	moded4	sheet2!b1:f4	2	1	merge	
4	par	moded4	sheet2!b8	2	1		
5	par	moded4	sheet2!i1	2	1	clear	
6							

which is addressed by the command

```
$call "gdxxrw i=gdxse.gdx o=gdxxrwss.xls  index=writeindex!a2"
```

and shows how the merge and clear commands are entered.

16.6.2.3 Other Options

Options exist for the control of type of output generated by Gdxxrw and the actions that occur in the spreadsheet when Gdxxrw runs.

16.6.2.3.1 Tracing Options

There are two options controlling the tracing output that can be created by Gdxxrw. According to the GAMS document the options are as follows.

16.6.2.3.1.1 Log and Logappend

Specifies the filename of the file to which Gdxxrw write information about its performance. When omitted, this information will be written to the computer standard output location (usually the screen). When using Gdxxrw in a GAMS model that is executed from the IDE, the output will be written to the IDE process window. Using LogAppend will cause the information to be appended to the end of the named file, while Log will overwrite the file. The syntax is

```
Log = FileName
LogAppend = FileName
```

where the filename must be valid on the computer and by default will be placed in the working directory.

16.6.2.3.1.2 Trace

Controls the amount of amount of information on Gdxxrw performance written to the log file or process window. Naturally error messages always are copied to the log file. The syntax is

```
Trace = integer
```

© 2022 Prof. Bruce McCarl

where the eligible integer values and the effect on Gdxxrw output are

- 0 Minimal Gdxxrw information is included in the output (Not even indications of errors).
- 1 Message appears telling about each Gdxxrw call indicating input file, output file and execution time (the default).
- 2 Message appears giving the level 1 output plus workbook name, and ranges worked with.
- 3 Message appears giving the level 2 output plus cell addresses, and numerical or string values for every item worked with.

The default value is 1.

16.6.2.3.2 Workbook performance options

There are two options controlling what happens in a workbook when Gdxxrw activates it. According to the GAMS document the options are as follows.

Updlinks

RunMacros

16.6.2.3.2.1 Updlinks

Specifies how links in a spreadsheet should be updated and is specified using

```
UpdLinks = integer
```

where the eligible integer values and the effect on spreadsheet actions are

- 0 Doesn't update any references (the default)
- 1 Updates external references but not remote references
- 2 Updates remote references but not external references
- 3 Updates both remote and external references

16.6.2.3.2.2 RunMacros

Controls the execution of automatic macros when the spreadsheet is opened and closed during the Gdxxrw operations. The macros involved are the 'Auto_open' and the 'Auto_close' macros. This is specified using

```
RunMacros = integer
```

where the eligible integer values and the effect on spreadsheet actions are

- 0 Doesn't execute any macros (default)
- 1 Executes Auto_open macro

© 2022 Prof. Bruce McCarl

2 Executes Auto_close macro

3 Executes Auto_open and Auto_close macro

16.6.2.3.3 Other GDXXRW Options

A number of other options are available in GDXXRW including

- RWAIT parameter that specifies a delay when opening Excel to avoid not ready problems
- A CheckDate option that will only regenerate output only if the input is more recent than output file
- `Password` that couses GAMS to pass a password when opening a protected Excel file

These and a few others are explained in gdxxrw .

16.6.2.4 Debugging Gdxxrw instructions

Just like anywhere else users will make mistakes in specifying Gdxxrw input. Such mistakes cause error messages in the log file or process window. Several procedures are appropriate when debugging a set of instructions

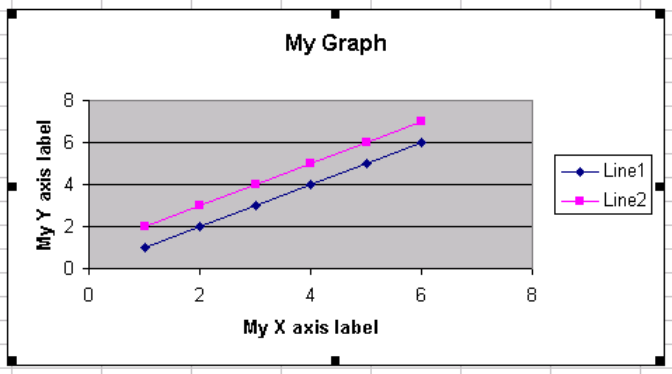
- When a new Gdxxrw is being set up users should use a trace value of 2 and then examine the log file or process window contents to make sure proper actions are being pursued.
- Start simple just getting one or two objects, not all that might be desired and work your way up.
- Note that when Gdxxrw encounters errors and is reading it does not generate a proper GDX file and subsequent procedures using that file may encounter errors.
- Be careful in only specifying upper left-hand corners of ranges
  - When reading always use this with a setting of skipempty=0.
  - When writing recognize the program will clear the entire sheet to the right and below the corner.

16.6.3 Spreadsheet graphics

May users of procedures such as Gnuplot and gnuplotxyz find themselves wanting more control over the graphics such as provided by Excel among other packages. One can use the Gdxxrw procedure to gain access to such graphics. In particular, one can set up a graph in Excel then use Gdxxrw to place new data in the spreadsheet in the area graphed and whenever the spreadsheet is subsequently opened the new and improved graph will be ready. Let me illustrate this using a new sheet called mygraph in our **gdxxrwss.xls**. Here I prepare a graph that uses the data in b2-d7 for the lines and the labels in a11 and a12 to describe those lines. I would have also liked to use the labels in a13-a15 to name the graph and label it's axes but could not figure a strategy to do this with Excel commands or a simple macro.

© 2022 Prof. Bruce McCarl

	A	B	C	D
1		x	Line1	Line2
2	i1	1	1	2
3	i2	2	2	3
4	i3	3	3	4
5	i4	4	4	5
6	i5	5	5	6
7	i6	6	6	7
8				
9				
10	X	Xdata		
11	Line1	Graph of Data set 1		
12	Line2	Graph of Data set 1		
13	Title	My graph of 2 lines		
14	XAXIS	My X axis label		
15	YAXIS	My Y axis label		
16				
17				



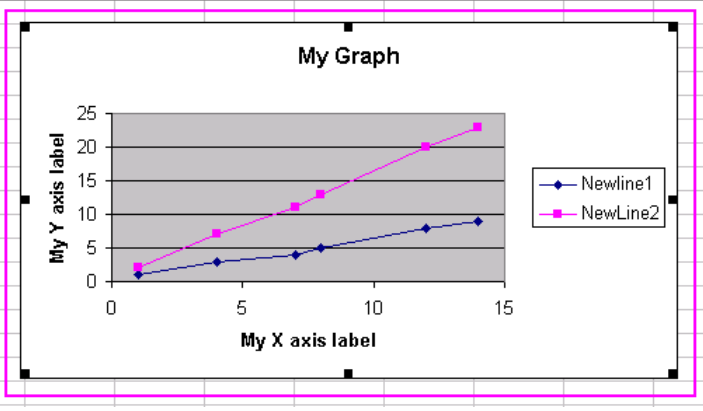
In turn I go into a GAMS program in this case **graphss.gms** and put in GDX file creation and Gdxxrw commands

```
set allels /X               X values
           Newline1         Data for line 1
           NewLine2         Data for line 2
           Title            A graph of 2 GAMS generated lines
           XAXIS            Whatever the X axis label
           YAXIS            Whatever the X axis label /
    points /i1*i6/;
table mydata(points,allels) data to graph
          x         NewLine1        NewLine2
i1        1           1
i2        4           3
i3        7           4
i4        8           5
i5        12          8
i6        14          9                 ;
mydata(points,"Newline2")=mydata(points,"x")+mydata(points,"Newline1");

execute_unload 'tograph.gdx' allels, mydata;
execute 'gdxxrw tograph.gdx trace=3 o=gdxxrwss.xls par=mydata Rng=mygraph!a1:d7
execute 'gdxxrw tograph.gdx trace=3 o=gdxxrwss.xls set=allels Rng=mygraph!a10 rd
```

where the Gdxxrw commands save the data to be graphed into the cells a1:d7 and a set using SET where I placed potential titles as explanatory text in cells a10 and below. In turn when I open the spreadsheet and the mygraph sheet as reproduced below where I find the data, legend and lines graphed have been derived from GAMS.

© 2022 Prof. Bruce McCarl

	A	B	C	D
1		X	Newline1	NewLine2
2	i1	1	1	2
3	i2	4	3	7
4	i3	7	4	11
5	i4	8	5	13
6	i5	12	8	20
7	i6	14	9	23
8				
9				
10	X	X values		
11	Newline1	Data for line 1		
12	NewLine2	Data for line 2		
13	Title	A graph of 2 GAMS generated lines		
14	XAXIS	Whatever the X axis label		
15	YAXIS	Whatever the X axis label		
16				
17				



16.6.4 Interactively including results

Many users desire to be able to execute another program during a GAMS job using some of the numbers resident in GAMS at that time then include the results in GAMS and continue onward. Unfortunately, GAMS has not yet released general interface routines for the GDX file usage from other programs so one needs to use the existing spreadsheet or GAMS mechanisms. One also can use an external program for constraint definition as discussed below.

16.6.4.1 Interactive calculations in a spreadsheet

The ability to use Gdxxrw to interact with a spreadsheet at execution time means that it is possible to have the spreadsheet do some things with data generated by GAMS and pass back the results interactively during a model run. In addition, there are certain processes that can be used in the spreadsheet that GAMS does not now contain. One such process is regression function estimation. Suppose a modeler wished estimate response functions across a group of scenarios run in a GAMS model. This can be done by interactively passing information to the spreadsheet, which in turn does the regression then collecting back the regression results. This can be done using a set of Gdxxrw, Execute_Unload, and Execute_Load commands.

Example:

Suppose I have set up a portfolio selection problem and solved it under alternative risk aversion parameters. Now I wish to estimate response functions for the individual stocks and the mean income as a function of the risk aversion parameter. This is done in the following GAMS code (**spreadinteract.gms**)

```
LOOP (RAPS,RAP=RISKAVER(RAPS);
            SOLVE EVPORTFOL USING NLP MAXIMIZING OBJ ;
            VAR = SUM(STOCK, SUM(STOCKS,
                INVEST.L(STOCK)*COVAR(STOCK,STOCKS)*INVEST.L(STOCKS))) ;
            OUTPUT("RAP",RAPS)=RAP;
            OUTPUT(STOCKS,RAPS)=INVEST.L(STOCKS);
            OUTPUT("OBJ",RAPS)=OBJ.L;
            OUTPUT("MEAN",RAPS)=SUM(STOCKS, MEAN(STOCKS) * INVEST.L(STOCKS));
            OUTPUT("VAR",RAPS) = VAR;
            OUTPUT("STD",RAPS)=SQRT(VAR);
            OUTPUT("SHADPRICE",RAPS)=INVESTAV.M;
```

© 2022 Prof. Bruce McCarl

```
                OUTPUT("IDLE",RAPS)=FUNDS-INVESTAV.L
                  );
 DISPLAY OUTPUT;
set funstoestimate(*);
funstoestimate("mean")=yes;
funstoestimate(stock)=yes;
set regpar /intercept,rap,rapsquare,rapcube,rapfour,rsquare/;
set regres /coef,stderr/
set rsqp /r2/
parameter rsq(rsqp);
PARAMETER estimatedata(RAPS,*) data to estimate regression over
PARAMETER regestimate(Regres,regpar) RESULTS FROM MODEL RUNS WITH VARYING RAP
PARAMETER regestimates(*,Regres,regpar) RESULTS FROM MODEL RUNS WITH VARYING RAI
loop(funstoestimate,
    estimatedata(raps,funstoestimate)=output(funstoestimate,raps);
    estimatedata(raps,'Intercept')=1;
    estimatedata(raps,'rap')=output('rap',raps);
    estimatedata(raps,'rapsquare')=output('rap',raps)**2;
    estimatedata(raps,'rapcube')=output('rap',raps)**3;
    estimatedata(raps,'rapfour')=output('rap',raps)**4;
    execute_unload 'regdata.gdx',estimatedata;
    execute 'gdxxrw regdata.gdx o=gdxxrwss.xls par=estimatedata Rng=regdata!a1'
    execute 'gdxxrw gdxxrwss.xls o=regdata.gdx par=regestimate Rng=regress!a1:f3
                                               par=rsq Rng=regress!a4:b4 rdim=1
    execute_load 'regdata.gdx',regestimate,rsq;
    regestimates(funstoestimate,Regres,regpar)=regestimate(Regres,regpar);
    regestimates(funstoestimate,'coef','rsquare')=rsq('r2');
    estimatedata(raps,funstoestimate)=0;
);
option regestimates:4:2:1;display regestimates;
```

where the portion in

- Red is running the GAMS model repeatedly for different risk aversion parameters.
- Blue is setting up the functions to estimate and the sets of information to pass back and forth and setting up the data to estimate a fourth order polynomial for the particular function to be estimated (as controlled by the loop on funstoestimate).
- Orange is unloading that data first to the regdata.gdx file.
- Purple places it into the spreadsheet **gdxxrwss.xls** on the regdata sheet that will automatically compute the regression results after the data are entered.
- Brown is loading the regression results from the spreadsheet into regress.gdx
- Pink takes the data from regdata.gdx file loading it into the GAMS program.

The spreadsheet **gdxxrwss.xls** sheets are

- regdata where the dependent variable goes into column b and the independent variables including an intercept goes into columns C-G as generated by the Gdxxrw command

© 2022 Prof. Bruce McCarl

```
execute 'gdxxrw regdata.gdx o=gdxxrwss.xls par=estimatedata Rng=regdata!a1
```

	A	B	C	D	E	F	G	H	I	J
1		BUYSTOCK4	intercept	rap	rapsquare	rapcube	rapfour			
2	R0		1							
3	R1		1	0.00025	6.25E-08	1.5625E-11	3.90625E-15			
4	R2		1	0.0005	2.5E-07	1.25E-10	6.25E-14			
5	R3		1	0.00075	5.63E-07	4.2188E-10	3.16406E-13			
6	R4		1	0.001	0.000001	1E-09	1E-12			
7	R5		1	0.0015	2.25E-06	3.375E-09	5.0625E-12			
8	R6		1	0.002	0.000004	8E-09	1.6E-11			
9	R7		1	0.003	0.000009	2.7E-08	8.1E-11			
10	R8		1	0.005	0.000025	1.25E-07	6.25E-10			
11	R9		1	0.01	0.0001	0.000001	0.00000001			
12	R10		1	0.011	0.000121	1.331E-06	1.4641E-08			
13	R11		1	0.0125	0.000156	1.9531E-06	2.44141E-08			
14	R12		1	0.015	0.000225	3.375E-06	5.0625E-08			
15	R13		1	0.025	0.000625	1.5625E-05	3.90625E-07			
16	R14	4.168303369	1	0.05	0.0025	0.000125	6.25E-06			
17	R15	6.828684873	1	0.1	0.01	0.001	0.0001			
18	R16	8.602272542	1	0.3	0.09	0.027	0.0081			
19	R17	8.956990091	1	0.5	0.25	0.125	0.0625			
20	R18	9.223028224	1	1	1	1	1			
21	R19	4.295667551	1	2.5	6.25	15.625	39.0625			
22	R20	2.147833776	1	5	25	125	625			
23	R21	1.073916888	1	10	100	1000	10000			
24	R22	0.715944592	1	15	225	3375	50625			
25	R23	0.536958444	1	20	400	8000	160000			
26	R24	0.268479222	1	40	1600	64000	2560000			
27	R25	0.134239611	1	80	6400	512000	40960000			
28										
29										
30										
31										

Sheet1 / output2 / writeindex / myindex / mygraph / Sheet2 / skipempty / output / solution / regress / **regdata**

- the regress one where the data are copied into cells L3:Q28 from the regdata sheet and the Excel function

```
=+LINEST(L3:L28,M3:Q28,FALSE,TRUE)
```

is used to perform the regression and place the results in the range b2:h6 with the labeling manually entered in row 1 and column A so it corresponds with the GAMS names.

	A	B	C	D	E	F	G	H	I
1		rapfour	rapcube	rapsquare	rap	intercept			
2	coef	-1.7436E-05	0.002358	-0.087382	0.804187	1.79437	0	#N/A	
3	stderr	1.50329E-05	0.002007	0.073578	0.734397	0.761572	#N/A	#N/A	
4	r2	0.091772381	3.245952	#N/A	#N/A	#N/A	#N/A	#N/A	
5		0.424391409	21	#N/A	#N/A	#N/A	#N/A	#N/A	
6		22.35737744	221.2603	#N/A	#N/A	#N/A	#N/A	#N/A	
7									

Sheet1 / output2 / writeindex / myindex / mygraph / Sheet2 / skipempty / output / solution / **regress**

In turn these data are loaded into a GDX file and on to GAMS using (note the Gdxxrw line below cannot be split into 2 lines when used but is here for readability)

© 2022 Prof. Bruce McCarl

```
execute 'gdxxrw gdxxrwss.xls o=regdata.gdx par=regestimate Rng=regress!a1
                                        par=rsq Rng=regress!a4:b4 rdim=1';
execute_load 'regdata.gdx',regestimate,rsq;
```

Finally the result is

```
----    119 PARAMETER regestimates  RESULTS FROM MODEL RUNS WITH VARYING RAP

                    intercept         rap    rapsquare     rapcube     rapfour

mean     .coef        133.7605    -27.6837       1.8079     -0.0414      0.0003
mean     .stderr        2.3321      2.2489       0.2253      0.0061 4.603474E-5
BUYSTOCK1.coef          1.2887      0.1672      -0.0239      0.0007 -5.13349E-6
BUYSTOCK1.stderr        0.4190      0.4041       0.0405      0.0011 8.270827E-6
BUYSTOCK2.coef          8.0474     -1.9637       0.1369     -0.0032 2.252415E-5
BUYSTOCK2.stderr        0.9416      0.9080       0.0910      0.0025 1.858634E-5
BUYSTOCK3.coef          6.7634     -2.1940       0.1670     -0.0041 2.885521E-5
BUYSTOCK3.stderr        1.2139      1.1706       0.1173      0.0032 2.396214E-5
BUYSTOCK4.coef          1.7944      0.8042      -0.0874      0.0024 -1.74360E-5
BUYSTOCK4.stderr        0.7616      0.7344       0.0736      0.0020 1.503290E-5
```

where the spreadsheet was used to do 5 regressions.

16.6.4.2 Calling GAMS from GAMS

Instances certainly arise where one wishes to include information from external programs into the current GAMS program. This cannot be done with $Include commands unless one uses save and restart since the $Include action only works at compile time and can only include a file that existed before the GAMS program began. Furthermore GAMS has not yet released general interface routines that can be called within programs written in other languages allowing one to extract and rewrite GDX files. However a trick may be employed to get around this where

- Data are saved using put for external file consumption
- The external program is executed generating a potential include file
- GAMS is executed from within the main GAMS program where in the subservient GAMS program
  - $include is used to bring in the include file data
  - Execute_unload is used to place those data in a GDX file
- The main GAMS program loads those data using Execute_Load.

Example:

Suppose for example I wish to invert a matrix and have a Delphi program (**invert1.exe**) for that. The first GAMS program (**invert.gms**) is as follows

```
set i /i1*i4 /;
alias(i,j);
table a(i,i)
       i1     i2    i3    i4
```

© 2022 Prof. Bruce McCarl

```
i1      2      0      2    1
i2      0      1      1    1
i3      0      0      1    3
i4      1      0      0    1
file mymatrix;
put mymatrix;
mymatrix.pc=5;
put '';loop(j,put j.tl;); put /;
loop(i,put i.tl;loop(j,put a(i,j));put /);
putclose;
execute_unload 'mygdx.gdx',i
execute 'invert1 i=mymatrix.put o=myinverse.put';
execute 'gams inverse2';
parameter ainv(i,j)
execute_load 'mygdx.gdx', ainv;
display ainv;
```

This programs proceeds through several steps as color coded to the statements above.

- Writes a put file to pass the matrix to be inverted to the external program.
- Write a GDX file that contains the sets needed by the second GAMS program.
- Executes the external matrix inversion program (invert1.exe) which in turn generates a file in GAMS format which contains the inverse.
- Executes another GAMS program which will incorporate the file containing the inverse and will write a GDX file as will be shown below.
- Read the inverse from the GDX file with execute_load.

Simultaneously in the second GAMS program (**inverse2.gms**) I

- Load in the sets I need from the main GAMS program.
- Include the file with the inverse as generated by the external inverter program.
- Save the inverse in a GDX file for inclusion in the original program.

```
$gdxin mygdx.gdx
set i;
$load i
$gdxin
alias(i,j);
table ainv(i,j)
$include myinverse.put
display ainv;
execute_unload 'mygdx.gdx', ainv;
```

Finally in the Delphi program which is in the archive **invert.zip** I have three basic segments

- One that reads the information form GAMS,

© 2022 Prof. Bruce McCarl

- One that does the work of the program inverting the matrix and
- One that writes the information back to GAMS.

Notes:

- This program allows one to include execution time information into GAMS using an $Include which is not ordinarily allowed. This is achieved by doing an execution time invocation of GAMS from within GAMS. This allows the second instance of GAMS to work with any files that have been created up to that point including the inverter created file with the inverse.
- In turn then having the embedded GAMS program write to a GDX file and using Execute_Load in the original program allows the information to be put back in at execution time.

16.7 Using equations defined by external programs

GAMS permits one to supply model constraint equations using an external custom written program. Here I will introduce the topic but note that there are a number of programming issues that are beyond the scope of this manual. Users wishing more details should consult the External Equations in GAMS section of the GAMS Interfaces Wiki. A more technical discussion is available in the External Functions document.

Identifying the equations and their contents: =X=
Building the external function evaluator

16.7.1 Identifying the equations and their contents: =X=

Inside GAMS one must tell which equations are to come from the external program. This is done using a .. specification with an equation relation type of =X=. Thus when one wishes to specify an equation though an external program one would specify some of the model equations in a manner as follows

```
zdefX .. sum(i, ord(i)*x(i) ) + (card(i)+1)* z =X= 1;
```

There are special conventions that must be followed in writing ..statements that involve =X= relationships.

- During the solution of models with external functions GAS will prepare and pass a vector of solution variables to a user written external function evaluator. The variables in this vector and the order in which they appear are controlled by the user and is specified during the writing of the .. =X= relations. In particular the **coefficients** for the variables in the .. =X= statements give the **position of the variable** in the vector.
  - This means that if multiple equations are to be defined by the external function evaluator that one must insure that each variable is always associated with the same number. Thus the variable x("setindex") must have the same exact integer coefficient "multiplying it" in every =X= equation in which it appears.
    - The mapping between GAMS variables and external variable indices must be one-to-one. This means that two GAMS columns cannot be mapped into the same external variable index.
    - The external variable indices must be contiguous from one to n (the total number of

© 2022 Prof. Bruce McCarl

external variables). There can be no holes in the list of external variable indices.

- These coefficients play no direct numerical role in the function evaluation. They just tell what variables are involved in what function and what the position is of the variables in the passed vector.
- These coefficients must be integer numbers.
- The true function of the variables is not required to be linear and can be of any allowable nonlinear form with smooth and continuous first derivatives but cannot involve any other named GAMS variables beyond those specified with coefficients.
- Thus in the problem above the x(i) will occupy positions 1 through n where n is the number of elements in i and z is in position n+1.
- The ordering of the variables in the vector is completely specified by the coefficients and does not depend at all on the internal ordering of GAMS variables and set elements. However, if one calculates the position using something like the ord function then the GAMS internal set order will control the order of the elements.
- One must carefully synchronize the GAMS program and the custom written function evaluation program so that when the 27^{th} element of the solution vector is addressed both the GAMS program and external program are dealing with the same variable.

- Multiple equations can be specified. The rhs specified in the =X= equation actually gives the number of the equation being specified. Thus in the specification

```
eqnum(k)=103+ord(k);
exteq(k).. sum(j,varnum(j)*x(j))=e=eqnum(k);
```

the equation associated with the first element of k would correspond to an equation number of 104.

- The actual function is interpreted as an equality constraint with a zero rhs.
  - Thus slack variables must be introduced for inequalities.
  - Any constants needed should either be embedded in the external evaluator or the equation decomposed. For example suppose I wished to include the equation

```
f(x)+g(x)-k1=k2
```

where f(x) is needed from the external program but the other terms can be kept in the GAMS program. This can be done by adding a new variable ZZ and specifying two equations

```
ZZ-f(x)=0
ZZ+g(x)-k1=k2
```

where the first equation would then be specified as an =X= equation and the other as a standard GAMS =e= equation.

Finally the model must be solved with an NLP capable solver.

© 2022 Prof. Bruce McCarl

16.7.2 Building the external function evaluator

The user must write a custom DLL to evaluate the functions. It must be callable as the function

```
GEFUNC (icntr, x, f, d, msgcb)
```

This code is created in a programming language (C, Delphi, Fortran etc.).

The GEFUNC parameters are

icntr	is used to communicate control information between the solver and the DLL and includes the number of the equation to be evaluated
x	is vector of variable values passed from GAMS to the solver
f	is evaluated value of the external function value at point x
d	is a vector containing derivatives of each variable in the function evaluated at the point x
msgcb	is a parameter that allows messages to be passed

Also the function must return a control code indicating whether any problems were encountered. In doing this care must be taken to insure that the derivatives and functions are continuous.

Notes:

- The DLL must generally have the extension DLL (note other extensions are used on non windows platforms).
- The DLL must generally be named with the name of the model solved but one can alter this using a file statement and including the named file in the Model statement as follows

```
file mydll /targetname.dll/;
model m /eq1,eq2,mydll/;
```

  where targetname.dll becomes the active name for the DLL.

Example:

(external.gms, extern.zip)

Suppose I use an example from the GAMS web page. Namely suppose I wish to specify the objective function of a quadratic model via an external function. In such a case one would define the objective function as follows (**external.gms**)

```
zdefX .. sum(i, ord(i)*x(i) ) + (card(i)+1)* z =X= 1;
```

and would also define an external DLL which evaluates this function. A Delphi version of this is in **extern.zip**.

17 Controlling GAMS from External Programs

When a modeler wants to link GAMS results or input to other programs it can be done in several fundamentally different ways.

- GAMS is in charge and data from other programs is to be incorporated into the GAMS

© 2022 Prof. Bruce McCarl

program as it starts up.

- GAMS is in charge and data from the GAMS results are to be passed to other programs at the conclusion of the GAMS run.
- GAMS is in charge and the user wants to run another program during a GAMS run.
- GAMS is in charge and the user wishes to pass data interactively to other programs interactively during a run.
- Equations in the user model are defined by an external program.
- Some other program is in charge and the user wants to use GAMS to solve a model.
- A GAMS model needs to be converted to another language for solution.

The first five of these are discussed in the chapter Links to Other Programs Including Spreadsheets, the other two are discussed herein.

Calling GAMS from other programs
Transferring models to other systems

17.1 Calling GAMS from other programs

Most of the time GAMS is in charge. However, GAMS can be in the background. In such a setting another program needs to activate GAMS, provide data, wait until GAMS is done and receive back solution information. An infinite number of variants with many other programs in charge are possible. Here I will provide 2 examples

- Excel spreadsheet in charge through Visual Basic Macros
- Compiled program in charge, in this case Delphi

A number of other variants including C++, Java, Visual Basic, and a web server are covered in section Executing GAMS from other Environments of the GAMS Interfaces Wiki.

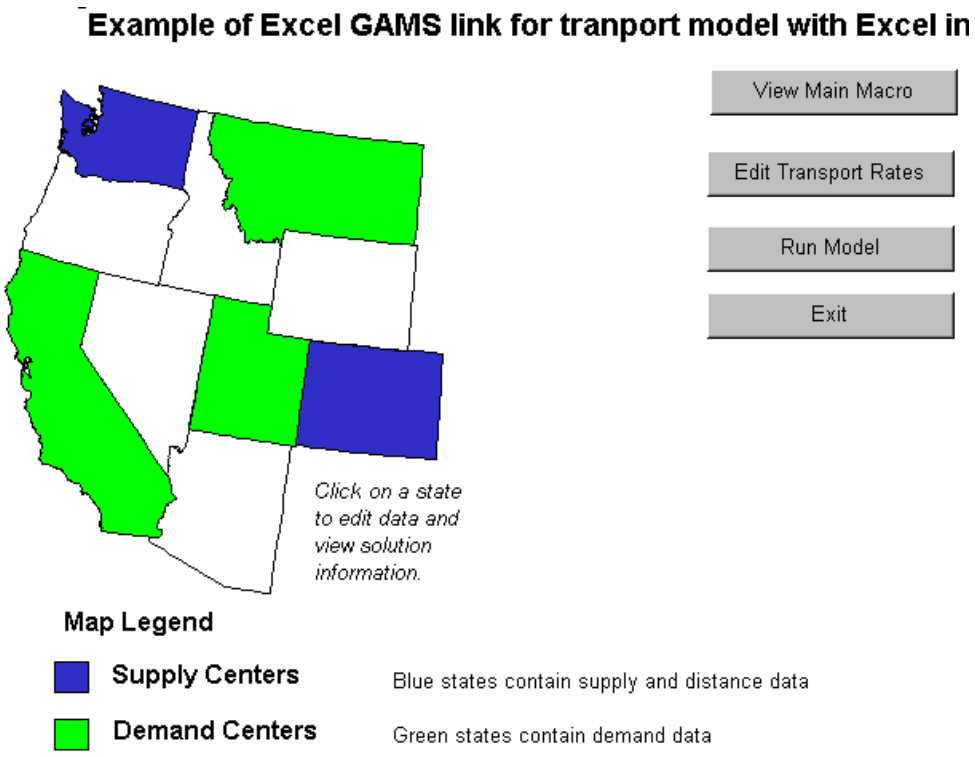
Excel spreadsheet in charge
Compiled program in charge – Delphi
Web servers or programs in other languages in charge

17.1.1 Excel spreadsheet in charge

One way GAMS can be used is as a slave to Excel that solves a predefined problem and facilitates data transfers. A little history is in order before showing exactly how this works. A number of years ago GAMS Corporation had a section on their web site displaying linkages and an early version of the Excel/ GAMS linkage program was there. However, I was unaware that it was a GAMS / Excel interface. Later Rob Davis with the Bureau of Reclamation came to one of my classes and told me he had been using an interface based on that GAMS program. I then looked into it and in the process reprogrammed the Excel macros slightly to improve the process and separate out the code functions to facilitate use by others. I also received assistance from Erwin Kalvelagen and Paul van der Eijk at GAMS Corporation in terms of finding the path for the GAMS system by reading the gamside.ini file. Finally, recent efforts led to me updating to use the Gdxxrw interface and Tanveer Butt assisted in that effort. So below I discuss an Excel / GAMS interface which is a product of many.

© 2022 Prof. Bruce McCarl

The base application involves a spreadsheet (**excelincharge.xls**) built around a transportation model implemented in the GAMS program **excelincharge.gms**. The application is centered around a map. The first page of the spreadsheet (Mapsheet) appears as follows



In this worksheet pressing each of the colored state maps transfers focus to a worksheet associated with state. Also pressing the buttons on the right causes macros to run which place you in the Visual Basic Editor, transfer you to the transport rate page, run GAMS, or exit.

17.1.1.1 Excel part of implementation

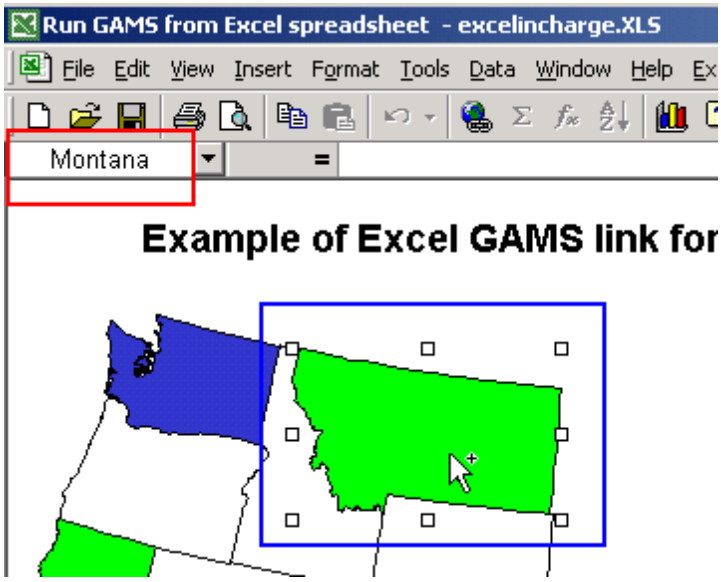
Now let us look at some details regarding the spreadsheet part of the implementation.

17.1.1.1.1 Defining the links through the map

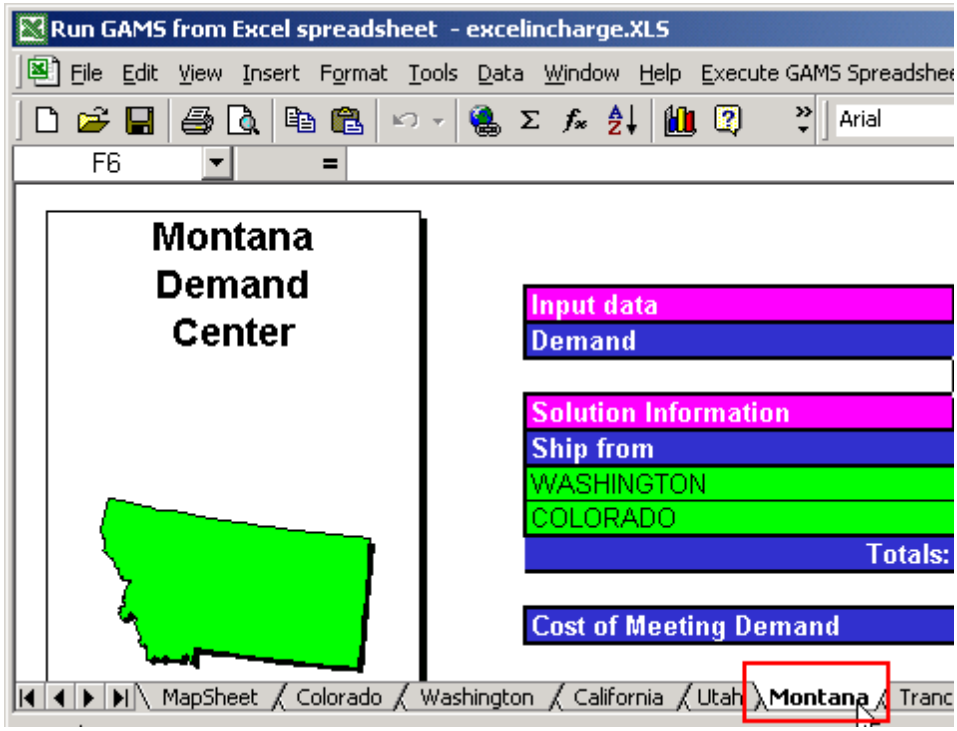
The main function of this application is controlled by the Mapsheet worksheet. The map on that worksheet is actually a collection of objects. It was originally defined in a package like Arcview, and then imported into Powerpoint and in turn Excel. During that process that map was ungrouped so that it was composed of individual objects for each state and each state was manipulated to allow the functionality required. In particular

- the state object was right clicked on (as in the case of Montana below) and given a name in the spreadsheet range name box as illustrated in the red square below

© 2022 Prof. Bruce McCarl

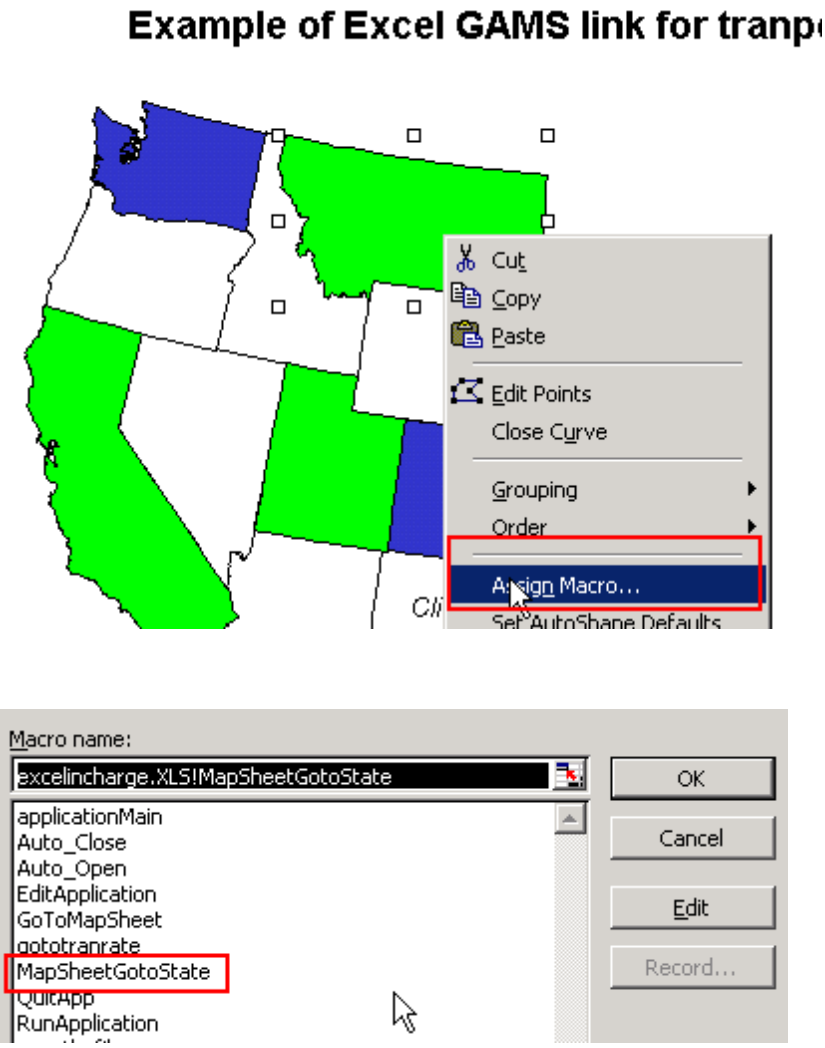


- A worksheet page was defined within the workbook named with just the same spelling as the state name



- The map object for each state that was to be active (in terms of transferring to a web page) was right clicked on and associated with the macro called **MapSheetGotoState** which uses the range name for the item clicked on to determine the sheet that will be displayed (thus the requirement that the sheet name and the range name be the same).

© 2022 Prof. Bruce McCarl



Note the state object Idaho was not associated with a state, nor was a worksheet added for it since it was not an active part of the model.

- In turn the worksheet with the name of the state was set up to have appropriate content.

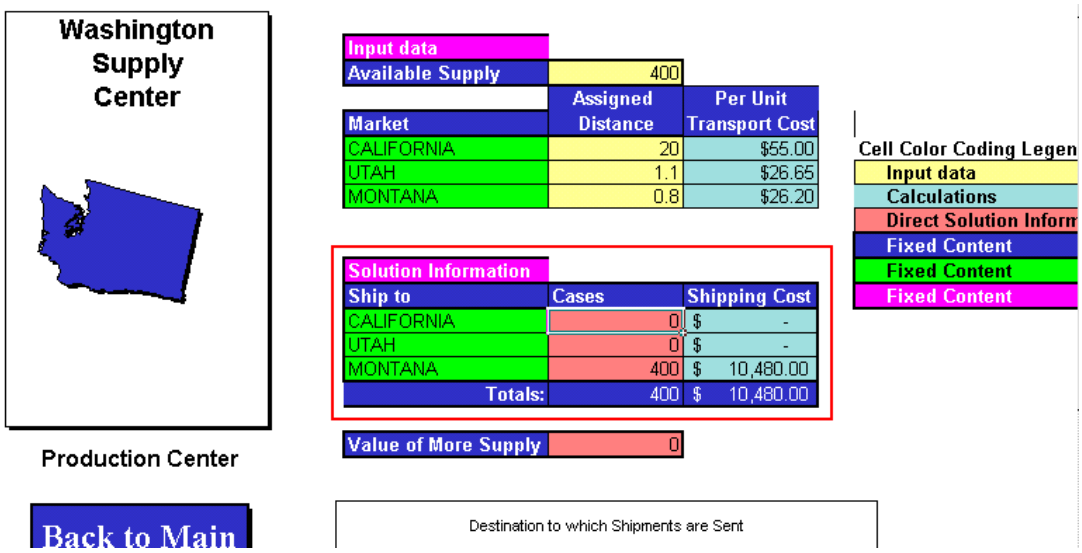
17.1.1.1.2 Worksheets present

In addition to the main Mapsheet the program contains

- An individual worksheet for each of the 5 states.
- A worksheet for transportation cost data.
- A sheet called Inputs that transfers data from other sheets so it can be read with Gdxxrw along as discussed below with an Index set of Gdxxrw instructions for reading.
- A sheet called Results that is addressed by other sheets when they require solution information that is the place Gdxxrw transfers data into along with an Index set of Gdxxrw instructions for writing.

© 2022 Prof. Bruce McCarl

These sheets are interconnected. For example pressing on the Mapsheet picture of the state of Washington brings up the sheet



that contains a mixture of solution information and raw data. The solution information in the red box and is transferred by formula from the results sheet (which contains information passed back from GAMS) as in cell F14 where the formula is

```
=Results!C2
```

The input data in the blue box once entered is transferred to the inputs sheet that in turn will be read by Gdxxrw by formula. For example in cell C21 of the input data sheet the formula

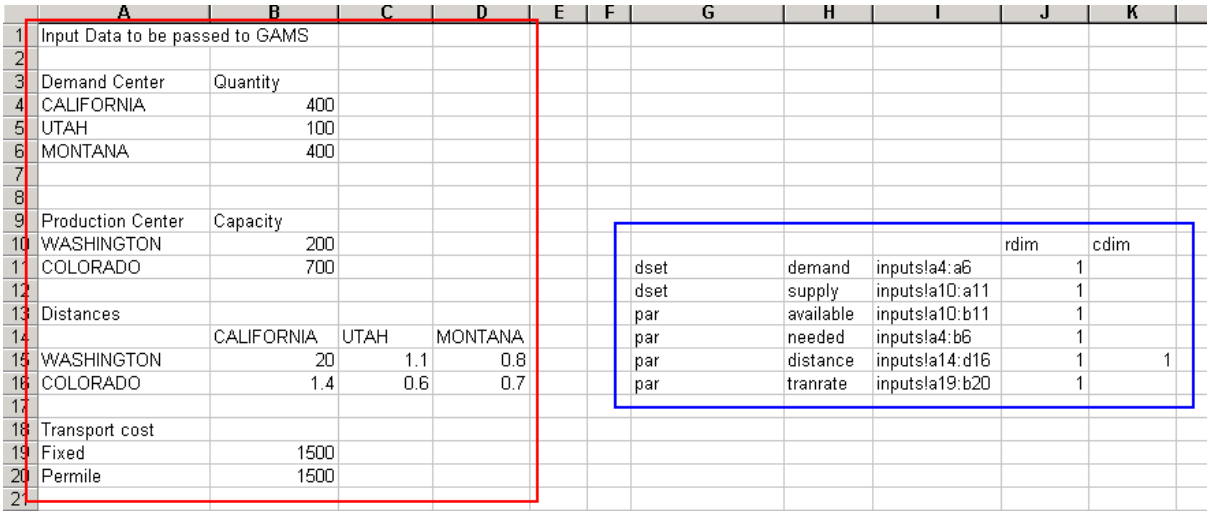
```
=wash_supply
```

appears where wash_supply is the range name for the cell F4 in the Washington sheet.

17.1.1.1.2.1 Inputs sheet structure

The inputs sheet is as follows

© 2022 Prof. Bruce McCarl



	A	B	C	D	E	F	G	H	I	J	K
1	Input Data to be passed to GAMS										
2											
3	Demand Center	Quantity									
4	CALIFORNIA	400									
5	UTAH	100									
6	MONTANA	400									
7											
8											
9	Production Center	Capacity									
10	WASHINGTON	200								rdim	cdim
11	COLORADO	700					dset	demand	inputs!a4:a6	1	
12							dset	supply	inputs!a10:a11	1	
13	Distances						par	available	inputs!a10:b11	1	
14		CALIFORNIA	UTAH	MONTANA			par	needed	inputs!a4:b6	1	
15	WASHINGTON	20	1.1	0.8			par	distance	inputs!a14:d16	1	1
16	COLORADO	1.4	0.6	0.7			par	tranrate	inputs!a19:b20	1	
17											
18	Transport cost										
19	Fixed	1500									
20	Permile	1500									
21											

where

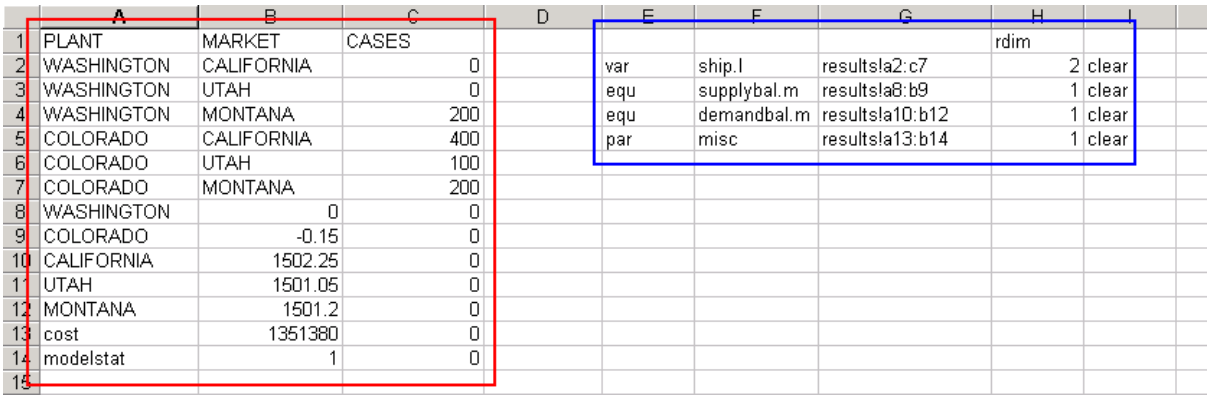
- The material in the red box will be read by Gdxxrw
- The worksheet will be saved so that the material read by Gdxxrw is current
- The material in the blue box is the set of commands telling Gdxxrw what types of items, item names, and ranges to read as covered in the Links to Other Programs Including Spreadsheets chapter using the index option where the input commands can be imbedded in a spreadsheet. This will be addressed in the **excelincharge.gms** file using the command

```
$Call 'Gdxxrw Excelincharge.XLS skipempty=0 trace=2 index=inputs!g10'
```

In addition all of the numbers in the red box are the results of formulas that copy the numbers in the state and trandata sheets.

17.1.1.1.2.2 Results sheet structure

The results sheet is as follows



	A	B	C	D	E	F	G	H	I
1	PLANT	MARKET	CASES					rdim	
2	WASHINGTON	CALIFORNIA	0		var	ship.l	results!a2:c7	2	clear
3	WASHINGTON	UTAH	0		equ	supplybal.m	results!a8:b9	1	clear
4	WASHINGTON	MONTANA	200		equ	demandbal.m	results!a10:b12	1	clear
5	COLORADO	CALIFORNIA	400		par	misc	results!a13:b14	1	clear
6	COLORADO	UTAH	100						
7	COLORADO	MONTANA	200						
8	WASHINGTON	0	0						
9	COLORADO	-0.15	0						
10	CALIFORNIA	1502.25	0						
11	UTAH	1501.05	0						
12	MONTANA	1501.2	0						
13	cost	1351380	0						
14	modelstat	1	0						
15									

where

- The material in the red box will be placed in this workssheet by Gdxxrw as guided by the commands in the blue box.
- The material in the blue box is the set of commands telling Gdxxrw what types of items, item names, and ranges to write into excelincharge.xls as covered in the Links to Other Programs Including Spreadsheets chapter using the index option where the commands are imbedded in the spreadsheet. This will be addressed in the **excelincharge.gms** file using the command.

```
Execute 'Gdxxrw Excelincharge.gdx skipempty=0 zeroout=0 trace=2 index=resu
```

- The worksheet is not shared so a trick is used to permit Gdxxrw to write to it. Namely the macro goes through the following steps
  - I. The spreadsheet is saved so the current data is on the disk.
  - II. The spreadsheet is saved as excelincharge.xls. This:
    - a. Closes the base spreadsheet making it ready for GDXXRW to write
    - b. Causes the screen to show excelincharge.xls while GAMS is solving causing the screen image to be preserved while GAMS is running
  - II. Runs GAMS which in turn uses Gdxxrw
  - III. Signals completion of the GAMS run
  - IV. Reopens the base spreadsheet excelincharge.xls which will now contain the solution passed by Gdxxrw and shows it on the screen
  - V. Closes excelincharge.xls

17.1.1.1.3 Running GAMS – the main macro

The basic operation of the program is that one uses the map and other sheets to get the data right and then presses the Run Model button. There is a macro associated with that button called trancode. There are several main sections in trancode.

17.1.1.1.3.1 Critical user defined items

At the top one finds the only line that most users may need to change in custom applications

```
' define gams model file name and listing file name
basename = "excelincharge"
```

which defines the root name of the associated gms file in this case **excelincharge.gms**.

17.1.1.1.3.2 GAMS run sequence

Further down in trancode is the section that runs GAMS. It consists of the lines below and is not likely to need to be changed by the user

```
    'save the file
        ActiveWorkbook.Save
ActiveWorkbook.SaveAs Filename:=ThisWorkbook.Path & "\excelincharge.XLS", FileFo
```

© 2022 Prof. Bruce McCarl

```
xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:=False _
      , CreateBackup:=False
        savename = ActiveWorkbook.Name
     'run the GAMS job
       'define additional parameters to attach to GAMS Call
       params = ""
       'now run gams
       lGAMSRet = GAMSrun(params, sGAMSFile)
     'reload the file with the data passed from GAMS
      ActiveWorkbook.Save
```

This progresses through several steps

- The workbook is saved and remaned so that when GAMS runs it can read the latest input data from the inputs sheet (note Gdxxrw can read from an open workbook but addresses the data in the saved fiole on disk. Also Gdxxrw cannot write to an open workbook that is not shared but does so slowly. Thus temporarily rename the worksheet so the source worksheet can be written to.).
- GAMS is run using the GAMS solution Macro procedure gamsrun that is in the Visual Basic module also in the spreadsheet called rungams.
- The spreadsheet which now contains the data placed by GAMS into the worksheet is reloaded by executing a save command.

17.1.1.1.4 Actions involved with executing GAMS

When the Run Model button is pushed the steps in the section above are executed. The last of these causes a job to be run using windows multi-threading capabilities. The big concerns in such a run are

- Making sure the rest of the program waits until GAMS is done
- Being able to detect if GAMS operated properly and messaging the user on model status.

The macro coding takes care of these functions.

During the run of this macro users may see the lower bar on the screen show processes being invoked indicating GAMS is running. But mainly GAMS runs in the background and users will never know that it is being used with Excel being in charge.

17.1.1.1.5 Examining the macros

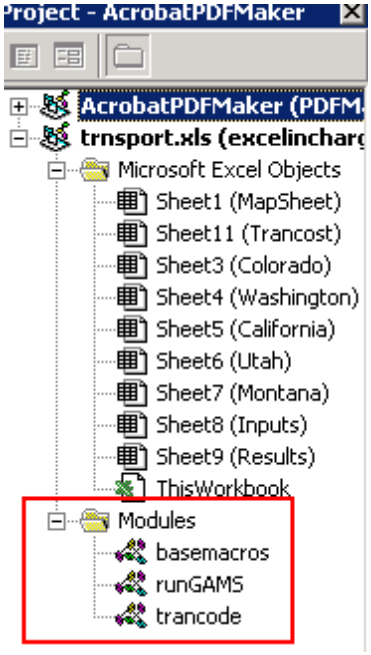
The **excelincharge.xls** workbook contains a number of macros. To see these one enters the workbook, presses the button

- Note Gdxxrw use for writing requires the workbook be shared or closed and sharing precludes inspection of the Visual Basic macros. Thus we do not used the shared option.

Once you have done this you will be placed in the trancode macro in the Visual Basic Editor. In the

© 2022 Prof. Bruce McCarl

window at the top left under the menu bar you will find a list of the Visual Basic modules which run this application.



There are 3 macro modules only one of which you ordinarily need be concerned with.

Visual Basic Segment	Code Purpose
trancode	Code that runs the transport application. The GAMS file name line at the top may need to be changed according to application
basemacro	Base macros starting up and managing the menu. This is likely generic but could change if one wished to do more with menu functions.
rungams	Code for running the GAMS job. The user should not change this.

17.1.1.2 GAMS part of implementation

There is also a base GAMS model underlying this application which is listed below

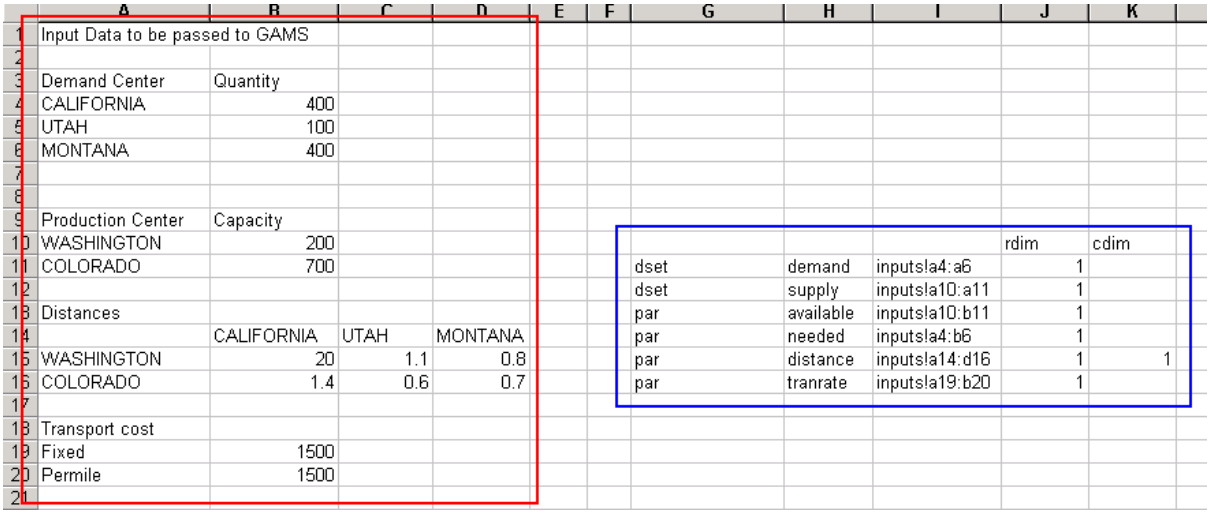
```
SETS   Supply     Locations of supply points
       Demand     Location of Demand markets;
$Call 'Gdxxrw Excelincharge.XLS skipempty=0 trace=2 index=inputs!g10'
$gdxin excelincharge.gdx
$Load supply demand
set tranparm parameters of transport rate function  /fixed, permile/
PARAMETERS   Available(supply)  Supply available in cases
```

© 2022 Prof. Bruce McCarl

```
            Needed(demand)  demand requirement in cases
            Distance(supply,demand)  distance in thousands of miles
            tranrate(tranparm) transport rate data;
$Load available needed distance tranrate
$gdxin
PARAMETER Cost(supply,demand)  transport cost in thousands of dollars per case ;
       Cost(supply,demand) = tranrate("Fixed")
                           + tranrate("permile") * Distance(supply,demand) / 10
positive VARIABLES    ship(supply,demand)  shipment quantities in cases
variable              Z       total transportation costs in thousands of dollars
EQUATIONS             COSTacct            define objective function
                      SUPPLYbal(supply)   observe supply limits at sources
                      DEMANDbal(demand)   satisfy demand requirements at markets
COSTacct ..        Z  =E=  SUM((supply,demand), Cost(supply,demand)*ship(supply,
SUPPLYbal(supply) ..   SUM(demand, ship(supply,demand))  =L=  Available(supply)
DEMANDbal(demand) ..   SUM(supply, ship(supply,demand))  =G=  needed(demand) ;
MODEL TRANSPORT /ALL/ ;
SOLVE TRANSPORT USING LP MINIMIZING Z ;
parameter misc(*);
misc("cost")=z.l;
misc("modelstat")=transport.modelstat;
Execute_Unload 'excelincharge.gdx',ship,  supplybal, demandbal, misc;
Execute 'Gdxxrw Excelincharge.gdx skipempty=0 zeroout=0 trace=2 index=results!e1
```

This code goes through several notable stages in terms of the Excel in charge application.

- When the code starts up Gdxxrw is called to transfer data for a number of items the just saved version of the **excelincharge.xls**spreadsheet into a GDX file. These include the element definitions for the supply and demand sets, along with the data for the available, needed, distance, and tranrate parameters.
- This transfer is controlled by the index field in the spreadsheet starting in g10 as in the blue box below.



	A	B	C	D	E	F	G	H	I	J	K
1	Input Data to be passed to GAMS										
2											
3	Demand Center	Quantity									
4	CALIFORNIA	400									
5	UTAH	100									
6	MONTANA	400									
7											
8											
9	Production Center	Capacity									
10	WASHINGTON	200								rdim	cdim
11	COLORADO	700					dset	demand	inputs!a4:a6	1	
12							dset	supply	inputs!a10:a11	1	
13	Distances						par	available	inputs!a10:b11	1	
14		CALIFORNIA	UTAH	MONTANA			par	needed	inputs!a4:b6	1	
15	WASHINGTON	20	1.1	0.8			par	distance	inputs!a14:d16	1	1
16	COLORADO	1.4	0.6	0.7			par	tranrate	inputs!a19:b20	1	
17											
18	Transport cost										
19	Fixed	1500									
20	Permile	1500									
21											

© 2022 Prof. Bruce McCarl

- Subsequently I load that data into sets and parameters that have been declared in GAMS.
- In these first two phases I use $Call and $Load rather than Execute and Execute_Load so that I may do compile time domain checking and since the data items are fully defined when the GAMS job starts. (See the discussion of the $Call and Execute choices in the chapter Links to Other Programs Including Spreadsheets and the discussion of $Load and Execute_Load in the Using GAMS Data Exchange or GDX Files chapter).
- Later I prepare to pass data back to the spreadsheet using a parameter array called Misc to pass assorted information including the objective function value and the model solution status indicator. That solution indicator is used in the Excel macros to see if the model solution is optimal in the Visual Basic component optstatus below that is part of the rungams macro. This code searches the results sheet first column for the word Modelstat which is the GAMS model solution status (this item is discussed in the Model Attributes chapter).

```
Function optstatus() As String
    Dim oResults As Range, oX As Range, nj As Integer, stat As Integer
    Set oResults = Worksheets("Results").Range("A1").CurrentRegion
     ' for each production center, update the results
    stat = 0
    For nj = 2 To oResults.Rows.Count
       If Trim(UCase(oResults.Cells(nj, 1))) = "MODELSTAT" Then
          stat = oResults.Cells(nj, 2)
       End If
    Next
    optstatus = "Unknown I cant find model stat"
    If stat > 0 Then
          Select Case stat
                Case 1
                     optstatus = "Optimal"
                Case 2
                     optstatus = "Optimal"
                Case 3
                     optstatus = "Unbounded"
                Case 4
                     optstatus = "Infeasible"
                Case Else
                     optstatus = "Bad Result from GAMS"
            End Select
    End If
End Function
```

- Next we unload data to a GDX file using Execute_Unload saving the ship variables, the demandbal and supplybal variables and the Misc parameter.
- Finally Gdxxrw is used to carry the information from the GDX file and place it into the **excelincharge.xls** spreadsheet. Note that spreadsheet needs to be shared at the time of this write.
- The information sent to the results worksheet is the optimal solution levels for the ship variable and the marginals from the supplybal and demandbal equations as well as the contents of the

© 2022 Prof. Bruce McCarl

Misc parameter as controlled by the index command and the spreadsheet range in the blue box below that starts at e1. The clear option is used in these commands to remove the old content and to assure the item order is the same as expected by the Excel workbook.

	A	B	C	D	E	F	G	H	I
1	PLANT	MARKET	CASES					rdim	
2	WASHINGTON	CALIFORNIA	0		var	ship.l	results!a2:c7	2	clear
3	WASHINGTON	UTAH	0		equ	supplybal.m	results!a8:b9	1	clear
4	WASHINGTON	MONTANA	200		equ	demandbal.m	results!a10:b12	1	clear
5	COLORADO	CALIFORNIA	400		par	misc	results!a13:b14	1	clear
6	COLORADO	UTAH	100						
7	COLORADO	MONTANA	200						
8	WASHINGTON	0	0						
9	COLORADO	-0.15	0						
10	CALIFORNIA	1502.25	0						
11	UTAH	1501.05	0						
12	MONTANA	1501.2	0						
13	cost	1351380	0						
14	modelstat	1	0						
15									

- In these last two phases I use Execute and Execute_Unload rather than $Call and $Unload so that I may do send the latest results from any solves and calculations. (See the discussion of the $Call and Execute choices in the chapter Links to Other Programs Including Spreadsheets and the discussion of $Unload and Execute_Unload in the Using GAMS Data Exchange or GDX Files chapter.

17.1.1.3 Developing Excel in charge – summary steps

There are a number of steps one needs to employ to make a application where Excel is in charge. The steps are as follows:

I. Set up a GAMS model that will eventually work with a spreadsheet at first ignoring the presence of a spreadsheet.

   a. Define all sets, data items etc just as you would for a stand-alone GAMS model. Such a model would be like **excelincharge1.gms** model which does not contain any

      - Gdxxrw $Call or Execute sequences
      - $Load / Execute_Unload commands

II. Develop a spreadsheet with an inputs sheet but just enter zeros for the results or omit them on the user pages (ie the state pages above).

   a. Format the inputs sheet so it collects all the data to be sent from the spreadsheet to the GAMS program from the use pages. Develop an index section in that sheet (as in **excelincharge.xls**) that will control Gdxxrw actions as discussed in the Links to Other Programs Including Spreadsheets chapter under the index section.

III. Redevelop the GAMS program so it uses Gdxxrw to read the input data. Use $Gdxin and $Load to import in the data items as in the top part of **excelincharge.gms**.

IV. Augment the spreadsheet so it contains a results sheet.

   a. Develop an index section in that sheet (as in **excelincharge.xls**) that will control Gdxxrw actions as discussed in the Links to Other Programs Including Spreadsheets chapter under the index section.

   b. In setting up the index section do not use clear at first. Rather get the information into

© 2022 Prof. Bruce McCarl

the sheet so its format matches what Gdxxrw will unload.

V. Use Gdxxrw to place the information in the Results sheet.

   a. After all the data have been satisfactorily unloaded switch to clear in the index section.

   b. Include an item that has Modelstat and the value of model stat in the A column of the spreadsheet.

VI. Make the spreadsheet application adding maps, other pages, sheet movement macros etc. (You will have to unshare the worksheet to work on the macros).

   a. Linking the inputs sheet to other sheets where input data items are added.

   b. Link the other sheets to the Results sheet to deliver solution information.

VII. Adapt the trancode macro so it will run your application. Change the file name of the GMS file in the critical user information section as discussed above.

17.1.2 Excel Spawning Alternative

An example called SpawnGAMSExcel (#84) is in the GAMS Data library that shows 3 different ways to specify the GAMS System Directory via VBA:

- By using the windows registry
- By using the "gamside.ini" file that is created when the GAMSIDE is opened
- Manually

Notes

- The GAMS model can contain several solve statements.
- The solution is obtained via the VBA code activation of "trace" command line parameter
- The VBA modules can be exported and used in other programs
- If the Excel Workbook is shared, then GDXXRW can be used to write the solution into the same Workbook

17.1.3 Compiled program in charge – Delphi

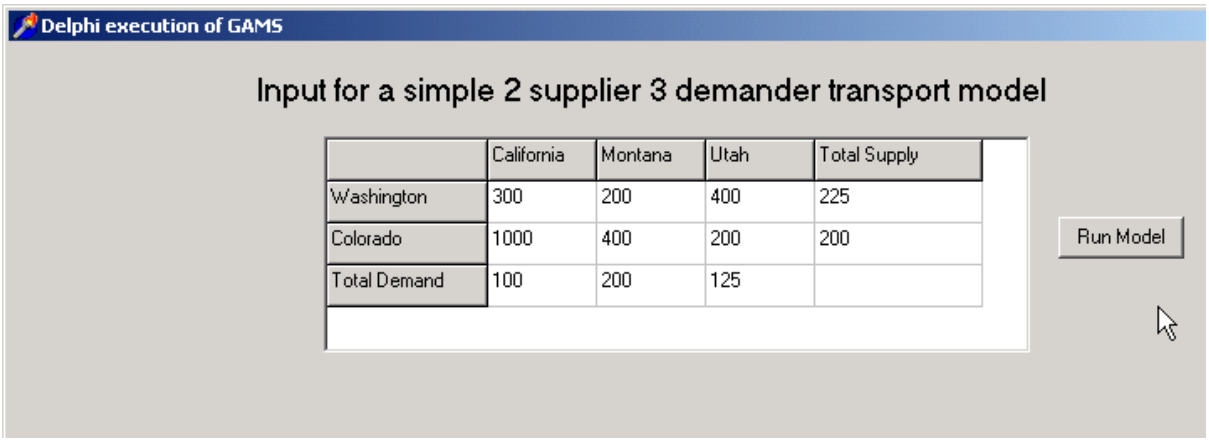
Yet another possibility is that a compiled program is to be in charge. Such an effort involves technical computing expertise that is outside the scope of the GAMS orientation of this document. As a consequence only a Delphi example will be pursued with a small discussion of the general issues. Those wishing to do such implementations in other languages should examine that below and the Visual Basic material above for general insight and look at the referenced in the section on other languages below.

17.1.3.1 A Delphi example

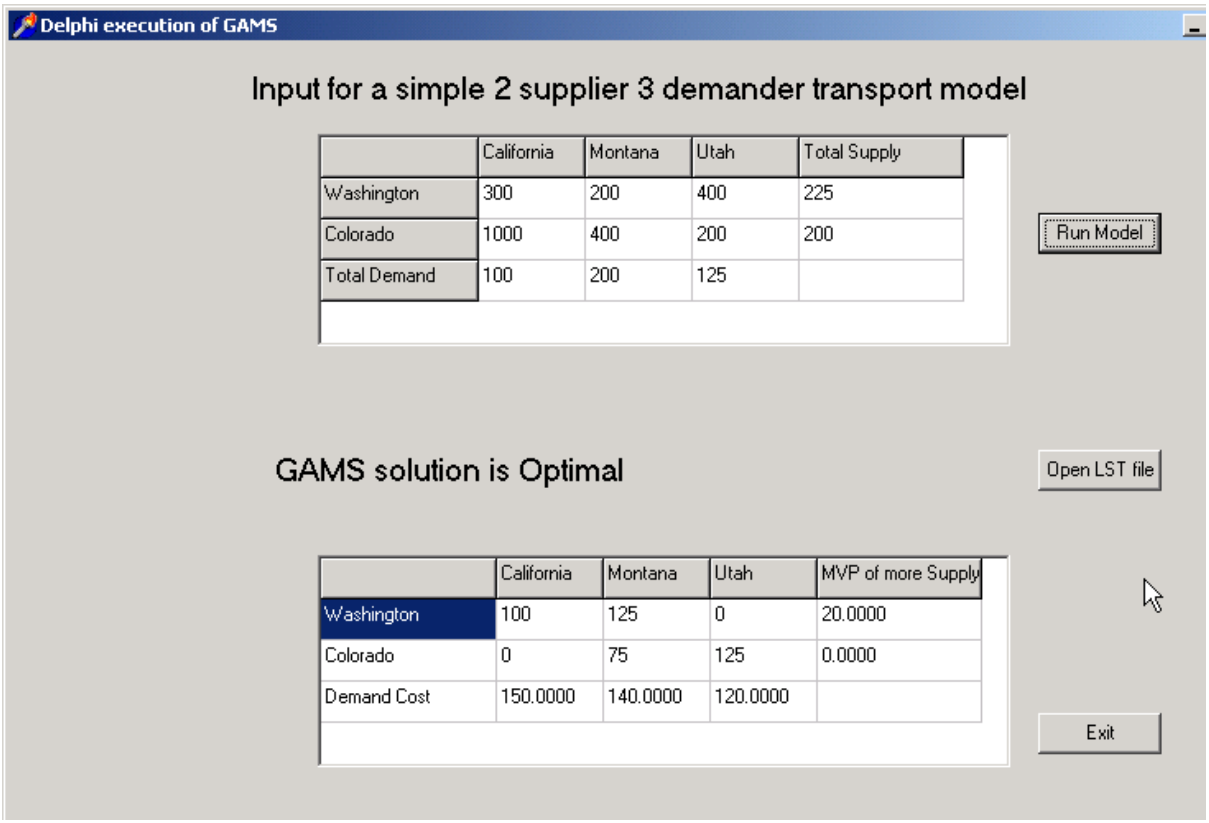
I have created a Delphi application which uses Kalvelagen's Delphi interface plus some other routines (all of that is in the module gamsmod.pas which is within the file **gamsrun.zip**). The problem is a transportation example, which accepts input and then on command runs the GAMS model, and displays

© 2022 Prof. Bruce McCarl

the answer. If you start up the file **gamsrun.exe** you will see the screen



This form that allows input for a simple transport model and is implemented in the module delphgam.pas which is within the file **gamsrun.zip**. After data are complete you can press the Run Model button. GAMS then executes and the screen is expanded with the solution as follows



17.1.3.1.1 Steps in application development

The Delphi implementation is similar in concept to the Visual Basic implementation and involves the steps

© 2022 Prof. Bruce McCarl

I. Develop the underlying GAMS model (**trandelp.gms**)

   a. Initially set it up as a stand alone model.

   b. Once the stand alone model works, then convert the program so its gets its data from CSV files (in the example these are named **supplyset.csv**, **demandset.csv**, **supplytbl.csv**, **demandtbl.csv**, **distancetbl.csv**).

   c. Develop a put file in CSV format that contains all the information one would want in the Delphi program (**output.csv**). Place the modelstat in the last row.

II. Develop a Delphi program which when told to run GAMS writes data to be passed to GAMS in CSV format which can also read CSV output from GAMS put files.

III. Develop the rest of the Delphi implementation to support the application.

The procedure is implemented in the gamsrun project in the delphgam.pas code and executes when the run button (button1) is pressed all of which is within the file **gamsrun.zip**.

17.1.3.1.2 Passing data to GAMS

In the Delphi program CSV files are created which contain the data to be passed. These CSV files are written by code such as

```
AssignFile(Fiwrite, 'distancetbl.csv');
Rewrite(fiwrite);
write(fiwrite,'fromdelphi');
for j:= 1 to 3 do
   write(fiwrite,',',stringgrid1.cells[j,0]);
writeln(fiwrite);
for i:= 1 to 2 do begin
   write(fiwrite,stringgrid1.cells[0,i]);
   for j:= 1 to 3 do
           write(fiwrite,',',stringgrid1.cells[j,i]);
   writeln(fiwrite);
end;
closefile(fiwrite);
```

which writes the distance matrix as stored in stringgrid1.

17.1.3.1.3 Calling GAMS

In the Delphi program GAMS is used via a set of routines embedded in gamsmod.pas. The basic GAMS execution is activated using the function ExecuteGAMS as follows

```
gamsfile:='trandelp.gms';
gamsparms:='';
returncode:=ExecuteGAMS(gamsfile,gamspath,gamsparms);
```

The status of the execution is obtained from another routine in gamsmod.pas as follows

```
strr:=GamsErrorString(returncode);
```

© 2022 Prof. Bruce McCarl

while the optimality etc status comes from reading the modelstat line at the end of the output.csv put file in the commands

```
readln(firead,strr);
k:=pos(',',strr);
strr:=copy(strr,k+1,length(strr));
stat:=strtoint(strr);
```

then calling a utility in gamsmod.pas which gives back a string on optimality status

```
strr:=optstatus(stat);
```

17.1.3.1.3.1 Challenges in running GAMS

One of the interesting problems one faces when developing code to run GAMS.EXE from within a compiled program in a Windows environment is multi-threading. If one does not take precautions, the program will proceed before GAMS is finished. Thus the code must wait until GAMS is finished. The module here as well as the macros in Excel illustrate how this can be done.

Another issue that needs to be addressed is GAMS need to store temporary files. By default this will be done in the current directory, a concept that is not always clear in a windowing environment. Herein the code uses the startup information that was present when **gamsrun.exe** was executed.

17.1.3.1.4 Reading the GAMS solution

In the Delphi program the results passed from GAMS are read by code such as

```
if FileExists('output.csv') then begin
   AssignFile(firead, 'output.csv');
   reset(firead);
   readln(firead,strr);
   for i:= 1 to 2 do begin
         for j:= 1 to 3 do begin
          readln(firead,strr);
          k:=pos(',',strr);
          strr:=copy(strr,k+1,length(strr));
          k:=pos(',',strr);
          strr:=copy(strr,k+1,length(strr));
          stringgrid2.cells[j,i]:=strr;
      end;
   end;
   for i:= 1 to 2 do begin
       readln(firead,strr);
       k:=pos(',',strr);
       strr:=copy(strr,k+1,length(strr));
       k:=pos('',strr);
       if(k>0) then  strr:=copy(strr,k+1,length(strr));
       stringgrid2.cells[4,i]:=strr;
```

© 2022 Prof. Bruce McCarl

```
end;
```

Which makes sure the file is present then reads and decodes the solution and places it in a Delphi feature called stringgrid2. The code above brings in the solution and supply shadow prices.

17.1.4 Web servers or programs in other languages in charge

Programs in virtually any programming language can be in charge. There are interface routines available in Visual Basic, Delphi, Visual C++, Oracle, and Java as covered in section Executing GAMS from other Environments .

There are also web server based implementations including

- GAMS-X developed by Tom Rutherford and Colin Starkweather which is described and available at *https://www.gams.com/latest/docs/UG_Tutorial.html*.
- GAMSsm developed by Tom Rutherford which is described and available at *http://www.mpsge.org/gamssm/index.html*.
- KESTREL developed by the NEOS group at *http://www-neos.mcs.anl.gov/neos/*. See also the KESTREL solver page.

These are all quite technical and those wishing go further are urged to consult the descriptions above along with technical language and operating system documents.

17.2 Transferring models to other systems

Users may wish to switch a model out of GAMS for solution elsewhere. This may be done using the solver called CONVERT as described in the document convert or via the *NEOS* procedures.

CONVERT transforms a GAMS model instance into a format used by other modeling and solutions systems. CONVERT is designed to achieve three aims:

- Permit users of GAMS to convert a confidential model into a GAMS solvable scalar form with very little identifying its structure so it can be given to a group for numerical investigation (i.e. to have someone else help with a solution problem while maintaining confidentiality).
- Give a path to solving with other solvers that may not be available in GAMS to test performance.
- Give a way of sharing test problems.

Currently, CONVERT can translate GAMS models into the following formats:

- AMPL
- BARON
- CPLEXLP
- CPLEXMPS
- GAMS
- LGO

© 2022 Prof. Bruce McCarl

- LINGO
- MINOPT

Models of the types LP, MIP, RMIP, NLP, MCP, MPEC, CNS, DNLP, RMINLP and MINLP can be converted.

The translator creates a "scalar model" which consists of

- A model without sets or indexed parameters in the scalar models, that is the modeled is transformed so it does not exploit the more advanced characteristics of any modeling system and is easily transformable.
- A model with a new set of individual variables depicting each variable in the GAMS model ending up with potentially 3 variable classes for the positive, integer, and binary variables each numbered sequentially (i.e. all positive GAMS variables are mapped into n single variables X1 - Xn thus if we have transport(i,j) and manufacture(k) we would have a set of unindexed scalar variables X1, X2, ... with i*j+k cases.).
- A model with individual equations depicting each variable in the GAMS model (ie all GAMS equations are mapped into m constraints E1, E2, ... Em thus if we have demand(j) and resources(r,k) we would have a new equations E1, E2,... with j+r*k cases).
- The symbolic form of these equations.
- Bounds and starting point values.
- A solve statement for the defined model.
- I should note that in this scalar conversion CONVERT removes most of the logic behind the structure of the problem making the model very difficult to understand, interpret and debug.

18 Utilities included in GAMS

Several types of utilities are included with GAMS. These include

- Posix file manipulation utilities
- Matrix manipulation utilities
- Interface and other utilities
- GDX related utilities

18.1 Posix utilities

The GAMS system for Windows includes a collection of Posix utilities generally for file manipulation. These utilities are typically available on Unix systems and therefore allow one to write platform independent file manipulation scripts, do file manipulations, compare files and do other things. The following utilities are available:

A A general purpose programming language that is designed for processing text-based data, either in files or data streams, and

© 2022 Prof. Bruce McCarl

was created at Bell Labs in the 1970s. AWK is described here.

C A command to concatenate and display files. CAT is described here.

C Reads files and calculates a checksum, cyclic redundancy check and the byte count for each. CKSUM is described here

C Compares two files of any type and writes the results to standard output. CMP results are empty if the files are the same; if they differ, the byte and line number at which the first difference occurred is reported. CMP is described here

C Reads two files as input, then outputs one file with three columns. The first two columns contain lines unique to the first and second file, respectively. The last column contains lines common to both. COMM is described here

C A utility that copies a file to a destination. CP is described here

C A utility which extract sections from lines of a file. Extraction of line segments can typically be done by bytes, characters, or fields. CUT is described here.

D A file comparison utility that outputs the differences between two files, or the changes made to a current file by comparing it to a former version of the same file. Diff displays the changes made per line for text files. DIFF is described here. One can also use the IDE through the Diff Textfiles feature.

E Evaluates a numerical expression and outputs the corresponding value. EXPR is described here

F Utility for breaking long lines into lines that have a maximum

© 2022 Prof. Bruce McCarl

width. FOLD is described here.

G Tells the current date and time and is a variant of the UNIX command DATE which is described here.

G Finds text within a file. GREP is described here.

G Sorts the lines in a text file. Renamed from the UNIX command SORT which is described here.

H Prints the first 10 lines of a file to the GAMS LOG file. The number of lines printed may be changed with a command line option. HEAD is described here.

J Merges the lines of two sorted text files based on the presence of a common field. JOIN is described here

M Moves files or directories from one place to another. When this is used the original file is deleted, and the new file may have the same or a different name. When the original and new files are in the same directory MV will rename the file instead. MOVE is described here

O Create an octal, hexadecimal or decimal dump of the data in a file. HEAD is described here

P Merges the lines of two files placing the contents of each line of a file at the end of the corresponding line of another file. PASTE is described here

© 2022 Prof. Bruce McCarl

P Takes an input string and writes it in a formatted fashion according to a formatting string. Bolded line, font color, numeric formats and other things can be specified. PRINTF is described here

R Deletes files or directories. RM is described here

S A batch executed file editor that follows prespecified commands to make changes to a text file. SED is described here

S A command that causes execution to pause for a specified amount of time. SLEEP is described here

T Prints the last 10 lines of a file to the GAMS LOG file. The number of lines printed may be changed with a command line option. TAIL is described here.

T Reads information from an input source and copies to the LOG file. Can be used to list all files that match a particular mask. TEE is described here.

T A command that tests to see if an expression is true. TEST is described here.

T Translate characters in a file to other characters using a user specified translation scheme generating a translated output file. TR is described here and here

U Reports on the incidence of or removes any adjacent duplicate lines in a file. UNIQ is described here.

© 2022 Prof. Bruce McCarl

W Counts words, lines, and or bytes in a file . WC is described here.

X Build and execute commands based on a file. XARGS is described here.

The collection consists of Windows versions of the GNU implementation of these utilities from Sourceforge. Detailed descriptions of the utilities can be found at the GNU website. The utilities "gdate" and "gsort" a renamed relative to the LINUX/UNIX commands of "date" and "sort" to avoid conflicts with the Windows commands "date" and "sort". For compatibility reasons the GNU implementation of awk called "gawk" has been renamed to "awk".

The diff utility can be used to compare files as in the example below or in the file filecompare.gms

```
*identify name of first file to compare
$setglobal file1 "tranerr.gms"
*identify name of second file to compare
$setglobal file2 "tranport.gms"
*show the control variables so you can check names are rifht
$show
*invoke the difference
$call "diff.exe %file1% %file2%"
*note the differences are shown in the LOG file
```

One can also ude the GAMSIDE Diff Textfiles option.

Examples can be found in the section GAMS Model Library under the files classified with application area GAMS Tools. The awk utility can be used to transform a variety of different text inputs into GAMS readable input files as illustrated in the model library file awkqap.gms.

18.2 Matrix Utilities

GAMS includes a set of matrix utilities. They are

INVERT	A routine that calculates the inverse of a matrix
CHOLESKY	A routine that does a Cholesky factorization of a symmetric matrix decomposition retuning a lower triangular matrix (L) such that the original matrix (A) equals the lower triangular matrix L times its transpose (A=LL')

© 2022 Prof. Bruce McCarl

EIGENVALUE	A routine that calculates eigenvalues of a symmetric matrix
EIGENVECTOR	A routine that calculates eigenvalues and eigenvectors of a symmetric matrix

All of these require use of a GDX file to first pass the data to the utility than to being back the result.

All make the assumption that the matrix to be worked on is of the form a(i,i) ie that the row and column index sets are the same. This is not typically the case for matrix inversion and an alternative inversion routine is given in inverse2.gms.

18.2.1 Invert

INVERT is a utility that calculates the inverse of a matrix

It is used by using $Call or Execute on the command

invert inputgdxfile indexofmatrix matrixtoinvert outputgdxfile resultantinverse

Where the parameters in this line are

inputgdxfile	Name of gdxfile sent to invert that has the matrix to be inverted
indexofmatrix	Name of set that defines row and column names of the matrix
matrixtoinvert	Name of the matrix to invert that must be in inputgdxfile
outputgdxfile	Name of gdxfile that has the resultant inverse matrix
resultantinverse	Name to call the inverse in the gdx file outputgdxfile

Example inverseexample.gms

```
Set index /i1*i2/;
Table a(index,index) matrix to invert
             i1   i2
    i1       2     1
    i2       1     3;
parameter ainverse(index,index) matrix that was inverted;
execute_unload 'gdxforinverse.gdx' index,a;
execute 'invert gdxforinverse.gdx index a gdxfrominverse.gdx ainverse';
```

© 2022 Prof. Bruce McCarl

```
execute_load 'gdxfrominverse.gdx' , ainverse;
display a, ainverse;
```

Users are likely to be constrained by the assumption that the matrix has the same column and row indices. Also having to define the GDX files every time may be inconvenient. Consequently a routine (arealinverter.gms) was written that provides a wrapper around the inverse it is called using the sequence

```
$batinclude arealinverter  matrixtoinvert rowindex colindex resultantinverse
```

Where the parameters in this line are

matrixtoinvert	Name of the matrix to invert that is of dimension (rowindex,colindex)
rowindex	Name of set that is the row index for the matrix to invert
colindex	Name of set that is the column index for the matrix to invert
resultantinverse	Array to receive the inverse that must be predefined and of dimension (colindex,rowindex)

Internally this procedure generates a matrix of the form required by INVERT and handles the GDX files.

Note the inverse is of opposite dimension order as compared with original matrix.

Example inverseexample.gms and

```
Set j /j1*j2/;
Table a2(index,j) second matrix to invert
            j1   j2
    i1       4     2
    i2       1     7;
Parameter a2inverse(j,index) inverse of second matrix
$batinclude arealinverter a2 index j a2inverse
display a2,a2inverse;
```

18.2.2 Cholesky

CHOLESKY is a utility that calculates the Cholesky decomposition of a symmetric positive-definite matrix. where given the matrix A one finds the matrix L such that A=LL'.

It is used by using $Call or Execute on the command

© 2022 Prof. Bruce McCarl

cholesky inputgdxfile indexofmatrix matrixtodecompose outputgdxfile resultantLmatrix

Where the parameters in this line are

inputgdxfile	Name of gdxfile sent to invert that has the matrix to be inverted
indexofmatrix	Name of set that defines row and column names of the matrix
matrixtodecompose	Name of the matrix to decompose that must be in inputgdxfile
outputgdxfile	Name of gdxfile that has the resultant L matrix
resultantinverse	Name to call the L matrix in the gdx file outputgdxfile

Example choleskyexample.gms

```
Set index /i1*i2/;
Table a(index,index) matrix to decompose
             i1   i2
    i1       2     1
    i2       1     3;
parameter LforA(index,index) L matrix that is a decomposition of A;
execute_unload 'gdxforutility.gdx' index,A;
execute 'cholesky gdxforutility.gdx index A gdxfromutility.gdx LforA';
execute_load 'gdxfromutility.gdx' , LforA;
display a, LforA;
```

18.2.3 Eigenvalue

EIGENVALUE is a utility that calculates the eigenvalues of a symmetric positive-definite matrix.

It is used by using $Call or Execute on the command

eigenvalue inputgdxfile indexofmatrix matrixtofindeigen outputgdxfile resultanteigenvalues

Where the parameters in this line are

inputgdxfile	Name of gdxfile sent to invert that has the matrix for which to find eigenvalues

© 2022 Prof. Bruce McCarl

indexofmatrix	Name of set that defines row and column names of the matrix
matrixtofindeigen	Name of the matrix to find eigenvalues for that must be in inputgdxfile
outputgdxfile	Name of gdxfile that has the resultant eigenvalues
resultanteigenvalues	Name to call the eigenvalue vector in the gdx file outputgdxfile

Example eigenexample.gms

```
Set index /i1*i2/;
Table a(index,index) matrix to find eigenvalues for
             i1   i2
    i1       2     1
    i2       1     3;
parameter eigenvalues(index) vector of eigenvalues of A;
execute_unload 'gdxforutility.gdx' index,A;
execute 'eigenvalue gdxforutility.gdx index A gdxfromutility.gdx eigenvalues';
execute_load 'gdxfromutility.gdx' , eigenvalues;
display a, eifenvalues;
```

18.2.4 Eigenvector

EIGENVECTOR is a utility that calculates the eigenvalues and eigenvectors of a symmetric positive-definite matrix.

It is used by using $Call or Execute on the command

eigenvector inputgdxfile indexofmatrix matrixtofindeigen outputgdxfile resultanteigenvalues resultanteigenvectors

Where the parameters in this line are

inputgdxfile	Name of gdxfile sent to invert that has the matrix for which to find eigenvalues
indexofmatrix	Name of set that defines row and column names of the matrix
matrixtofindeigen	Name of the matrix to find eigenvalues for that must be in inputgdxfile
outputgdxfile	Name of gdxfile that has the resultant eigenvalues

© 2022 Prof. Bruce McCarl

resultanteigenvalues	Name to call the eigenvalue vector in the gdx file outputgdxfile
resultanteigenvectors	Name to call the eigenvector matrix in the gdx file outputgdxfile

Example eigenexample.gms

```
Set index /i1*i2/;
Table a(index,index) matrix to find eigenvalues for
             i1   i2
    i1       2     1
    i2       1     3;
parameter eigenvalues(index) vector of eigenvalues of A;
parameter eigenvector(index,index) matrix of eigenvectors of A;
execute_unload 'gdxforutility.gdx' index,A;
execute 'eigenvector gdxforutility.gdx index A gdxfromutility.gdx eigenvalues ei
execute_load 'gdxfromutility.gdx' , eigenvalues, eigenvectors;
display a, eigenvalues,eigenvectors;
```

18.3 GDX Utilities

There are a number of GDX (GAMS Data Exchange) related utilities that are included with GAMS that work with GDX files.

These are

CSV2GDX	Converts a CSV file to a GDX file
GDXCOPY	Copies/Converts one or more GDX files to a different format
GDXDIFF	Inspects the data within symbols in two different GDX files that have the same name, type and dimension then writes any cases where differences are found into a third GDX file
GDXDUMP	Writes the contents of a GDX file into a GAMS formatted text file.
GDXMERGE	Combines the data from multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a items with the same symbol name with one additional dimension. The added dimension contains the file name of the file from which the data came from as the set element name.
GDXRANK	Reads 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 set elements in a GDX data-file using the revised names in a two dimensional set called map in a second GDX file. The GDX file is overwritten by

© 2022 Prof. Bruce McCarl

the command.

GDXVIEWER	Moves data from a GDX file and places it a CSV,XLS, XLS, GAMS ,Access, SQL, MS SQL, SQL Insert script, SQL Update script, HTML, or XML.
GDXXRW	Allows reading and writing of data from and to an Excel spreadsheet. This utility requires the presence of Microsoft Excel on the computer and therefore can only be used on a PC running the Windows operating system with Microsoft Excel installed
GDX2ACCESS	Dumps the contents of a GDX file to an MS Access file (.mdb file).
GDX2SQLITE	Dumps GDX contents into a SQLite database file (.db).
GDX2XLS	Places the contents of an entire GDX file into a Microsoft Excel spread sheet.
MCFILTER	Filters out dominated points
MDB2GMS	Takes contents from an Microsoft Access database into either a GAMS Include File or a GAMS GDX File.
SQL2GMS	Takes contents from an SQL database into either a GAMS Include File or a GAMS GDX file.
XLS2GMS	Takes contents from an Excel spreadsheet into either a GAMS include File or a GAMS GDX File.

- **ASK**: The ASK utility is a simple tool to ask simple interactive questions to the end-user. For instance, if your model requires a scalar to be changed regularly, instead of letting the end-user change the .gms source file, it may be better to pop up a window, with a question text, where the required number can be entered → Documentation.

- **ShellExecute**: ShellExecute is a small wrapper program for the shellexecute Windows API call. It allows you to spawn an external program based on the file type of the document to open → Documentation.

18.3.1 CSV2GDX

A utility that converts a CSV file to a GDX files called CSV2GDX.

Documentation is described here: csv2gdx

A simple example of its use is in the GAMS data library file called CSV2GDX1 (#85)

© 2022 Prof. Bruce McCarl

18.3.2 Gdx2sqlite

GDX2SQLITE is a Tool developed by Erwin Kalvelagen to dump GDX contents into a SQLite database file.

Usage:

gdx2sqlite -i inputfile -o outputfile {Options}

where

-i gdxinputfile Specifies the input GDX file. Typically this is a file with a .gdx extension

-o sqloutputfile Specifies the output SQLite database. Typically this file has a .db extension.

Options include

-debug	This is an optional flag that will cause gdx2sqlite to print additional debugging information.
-expltext	This optional flag will export explanatory text for set elements.
-append	Don't delete the database file before processing. This will allow adding new symbols in new tables. We will not allow adding data to existing tables.
-small	Write data strings in a separate table. A user-friendly SQL VIEW is created to hide the complexities of the joins.
-fast	Try to speed up writing the data using some non-standard pragmas. Using both -small -fast will write the data most efficiently.

Examples are provided here

Some notes

- GAMS does not store zero values and zero entries will not be exported to either the GDX file or on to the database. To force a zero to be exported, you need to manually set the values to EPS in GAMS. E.g. using p(i)$(p(i)=0) = EPS;
- N-dimensional sets are stored as tables with n text columns. In case the option -expltext is used, another column may be added with explanatory text.
- N-dimensional parameters will have n index columns plus a value column. Scalars are collected in a separate table.
- N-dimensional variables and equations have n index columns and also columns for the level, the lower and upper-bound and the marginal.

© 2022 Prof. Bruce McCarl

- Scalars are collected in the tables scalarvariables and scalarequations.
- The values INF and -INF are mapped to 1.0e100 and -1.0e100 in the database.
- The special value EPS is exported as zero.
- UNDEF, NA and acronyms are exported as NULL items.
- A database table is not allowed to have columns with the same name. If a name clash is detected new names may be invented.

More information also appears on Erwin Kalvelagen's blog
see http://yetanothermathprogrammingconsultant.blogspot.com/2014/06/big-data-cubes.html.
or http://yetanothermathprogrammingconsultant.blogspot.com/2014/07/a-little-bit-extra-fine-tuning.html.

18.3.3 Gdxcopy

The GDXCOPY utility provides a mechanism to convert GDX files to a prior format, so an older GAMS system can read these files. A current GAMS system can always handle older GDX file formats. For details see here.

18.3.4 Gdxdiff

GDXDIFF compares the data of for items in two GDX files and writes a GDX file showing the differences. For a description see the coverage here.

18.3.5 Gdxdump

GDXDUMP can take the symbols in a GDX file and write them into file that is formatted either

- As txt in GAMS readable commands that can be used in subsequent GAMS runs or
- As a CSV file.

The information that GDXDUMP will write in the output file is

- Data for a selected set, parameter, variable or equation (under all three of the output options when a specific item is named using the SYMB option)
- Data for all sets, parameters, variables and equations (Under *normal* option when the SYMB is not used)
- Data on solution items (variables and equations) formatted in a fashion suitable for import as a basis in another GAMS program using the GAMSBAS format where the marginals and levels are output (as discussed in the Expanded GAMS user guide under the heading *Advanced basis formation in GAMS* and also in the writeup for the now obsolete GAMSBAS solver in *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/papers/550.pdf*).

© 2022 Prof. Bruce McCarl

There are some peculiarities in the GDXDUMP. Namely

- Under the *CSV* format it only creates output when a symbol is selected using the SYMB syntax.
- Under the *CSV* format when the requested symbol is a variable or an equation one only gets the level values not the marginal, under the other formats one gets all items.
- Under the *gamsbas* format one gets all variables and equations when the SYMB syntax is **not** used.

More options are available and are described in the gdxdump writeup as accessible at GDXDUMP or through the GAMSIDE help under docs and tools

Usage is discussed here.

18.3.6 Gdxmerge

GDXMERGE combines the information from several GDX files and merges it into one composite file. Symbols with the same name, dimension and type that appear in the separate files are combined into a single symbol with an added dimension in the first index position that gives the file name. Its usage is discussed here.

18.3.7 Gdxrank

GDXRANK will sort all one dimensional parameters in a GDX file, and then write the sorted indices as one dimensional parameters in an output GDX file.

Usage:

gdxrank inputfile outputfile

Each one dimensional parameter will then be read from the input file, sorted and a corresponding integer permutation index will be written to the output file using the same name for the symbol. GAMS special values such as Eps, +Inf and -Inf are recognized.

Example gdxrank.gms

```
set I /i1 * i6/;
parameter A(I) /i1=+Inf, i2=-Inf, i3=Eps, i4= 10, i5=30, i6=20/;
display A;
* write symbol Array to sort to gdx file
execute_unload "rank_in.gdx", A;
* sort symbol; permutation index will be named A also
execute 'gdxrank rank_in.gdx rank_out.gdx';
* load the permutation index
parameter AIndex(i);
execute_load "rank_out.gdx", AIndex=A;
display AIndex;
```

© 2022 Prof. Bruce McCarl

18.3.8 Gdxrename

The gdxrename utility renames the text for set elements in a GDX data-file. The renaming follows the contents of a two dimensional set called map in a second GDX file.

The renaming of the set elements overwrites the string stored for each element, but does not change the data order for the data stored in the data-file. These changes are applied to the data-file directly overwriting its contents.

In the two dimensional set one defines a number of two dimensional tuples that in first position contains the original name and in the second position contains the new name. For example if one wants all the set elements called r renamed to red and all the ones called g to green one would have a two dimensional set called map in the named GDX mapfile that had tuple entries that look like

```
set map(*,*) /r.red, g.green/;
```

The replacement would occur in all sets, parameters etc where the specified set element names appear.

Usage:

gdxrename nameofgdxdatafile nameofgdxmapfile {Options}

where

nameofgdxdatafile is the name of the file where the renaming is to be applied

nameofgdxmapfile is the name of the file where the map set appears containing the renaming tuples

Options for now only includes **-R** or **-Reverse** which can be used to instruct Gdxrename to reverse the direction of the rename operation

Example (gdxcreateforrename.gms)

First, create a GDX file called **data.gdx** (gdxcreateforrename.gms) that the renaming will be applied to

```
set c /r, g, b, y/;
parameter A(c);
A(c) = Ord(c);
execute_unload 'data.gdx', A,c;
display a,c
```

Second, create a map set specifying the renaming rules and save it in a GDX file here using one named **map.gdx** (also in gdxcreateforrename.gms)

```
set map(*,*) /r.red, g.green, b.blue, y.yellow/;
```

© 2022 Prof. Bruce McCarl

```
execute_unload 'map.gdx' map;
```

Third, run GDXRENAME which will rename the set elements in a specified GDX file (here data.gdx) following the rules in the two dimensional set called map in the named GDX mapping here called **map.gdx** (also in gdxcreateforrename.gms)

```
execute 'gdxrename data map';
```

Fourth, load and display the results (gdxafterrename.gms);

```
set c;
$gdxin data.gdx
$load c
parameter a(c)
$load a
$gdxin
display a,c;
```

In turn before the renaming we had

```
----      7 PARAMETER A
      r 1.000,    g 2.000,    b 3.000,    y 4.000
----      7 SET c
      r,    g,    b,    y
```

afterwords we get

```
----      7 PARAMETER a
      red    1.000,    green  2.000,    blue   3.000,    yellow 4.000
----      7 SET c
      red   ,    green ,    blue  ,    yellow
```

18.3.9 Gdxrrw

GDXRRW is an interface between GAMS and R. It allows one to transfer data between GDX and R plus provides a way to call gams from within R.

Like the Excel interface GDXXRW, it is a data interface only with the model itself specified in GAMS. GDXRRW consists primarily of procedures that can read or write data into and out of GDX file. It also includes a function to cause GAMS to execute a model plus some utility functions.

GDXRRW is available in R source and binary form, which can be used in interactive R sessions and in R scripts. It follows the R conventions for building, examples, and documentation.

Documentation is available under the docs and utilities part of the IDE help menu plus at http://support.gams.com/doku.php?id=gdxrrw:interfacing_gams_and_r.

There are some examples of use at

https://www.wecr.wur.nl/gamstools/R%20for%20Gams%20Users.docx

© 2022 Prof. Bruce McCarl

18.3.10 Gdxviewer

`gdx2xls`: Converts an entire gdx data container to a Microsoft Excel spread sheet

GDXVIEWER moves data from a GDX file and place it a CSV,XLS, XLS, GAMS ,Access, SQL, MS SQL, SQL Insert script, SQL Update script, HTML, or XML. Gdxviewer is discussed here.

18.3.11 Gdxxrw

Utility that allows reading and writing of data from and to an Excel spreadsheet. This utility requires the presence of Microsoft Excel on the computer and therefore can only be used on a PC running the Windows operating system with Microsoft Excel installed. Its use is discussed here.

18.3.12 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.

18.3.13 Gdx2xls

GDX2XLS places the contents of an entire gdx file into a Microsoft Excel spread sheet. An overall index sheet is created and a separate sheet for each item in the file. The name of the spreadsheet is the gdx file root with the extension xls.

Example (gdx2xls.gms)

```
$call "gdx2xls gdxall.gdx"
```

Result

The result is a file named **gdxall.xls** with a first page as follows and note at the bottom separate sheets for each item in the GDX file**.**

© 2022 Prof. Bruce McCarl

Microsoft Excel - gdxall.xls

File Edit View Insert Format Tools Data Window

SnagIt Window

A1 'Name

	A	B	C	D	E
1	Name	Type	Dim	Count	Explanat
2	distance	parameter	2	8	
3	i1	set	1	3	
4	i10	set	2	4	
5	i10a	set	2	4	
6	i1a	set	1	3	
7	i3	set	1	5	
8	i4	set	1	6	
9	i4a	set	1	6	
10	i5	set	1	6	
11	i6	set	1	3	
12	i6a	set	1	3	
13	i6c	set	1	3	
14	i7	set	1	2	
15	i8	set	1	2	
16	i9	set	1	2	
17	j1	set	1	3	
18	j1a	set	1	3	
19	j2	set	1	4	
20	j3	set	1	5	
21	j4	set	1	3	
22	j5	set	1	2	
23	j6	set	1	2	
24	modedista	parameter	3	8	
25	modedista	parameter	3	6	
26	modedista	parameter	3	6	
27	modedista	parameter	3	6	
28					
29					
30					
31					
32					
33					
34					

Table of Contents / distance / i1 / i10 / i10a / i1a

18.3.14 MCFilter

A GDX utility that filters duplicate and dominated points from a solution set that is contained in a GDX file. To use this one specifies a array of points, a set of objectives evaluated at those points and a direction of preference for each objective. Cursory documentation occurs at the end of the gdxutils.pdf writeup as accessible at MCFILTER or through the GAMSIDE help under docs and tools

18.3.15 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. It is discussed here.

18.3.16 SQL2GMS

SQL2GMS transfers data from SQL compatible databases to GAMS. Its use is discussed here.

18.3.17 Xls2gms

Copies data from EXCEL to GAMS. It is discussed here.

18.4 Interface utilities

There are several utilities that permit communication with other programs or the user.

ASK	Allows one to ask simple interactive questions of the end-user.
MSAPPAVAIL	Checks which Microsoft Office programs are installed
SHELLEXECUTE	Allows one to spawn an external program based on the file type of the document to open
XLSTalk	Allows for some simple communication with Excel

. .

- :
- : ShellExecute is a small wrapper program for the shellexecute Windows API call. It allows you to spawn an external program based on the file type of the document to open → Documentation.
- : .

18.4.1 Ask

ASK is a utility developed by Erwin Kalvelagen to ask simple interactive questions of the end-user. For instance, if your model requires a scalar to be changed regularly, instead of letting the end-user change the .gms source file, it may be better to pop up a window, with a question text, where the

© 2022 Prof. Bruce McCarl

required number can be entered .

Ask is in the form of a GUI (Graphical User Interface). The main purpose of it is to allow a developer quickly put an application together such that an end user does not have to edit GAMS files.

It requires a GAMS model run in the GAMS-IDE and generates a standard GAMS include file, this file can then be used through a $include statement

Usage

ask <options>

where the options are

T= string	where the *string* identifies the type of input item to go after and can be *integer* when one wants an integer number *float* when one wants a real number *radiobutton* when one wants a radio button choice *combobox* when one wants a combo (drop down choice) box *checklistbox* when one wants a check list box *fileopenbox* when one wants the name of a file to open *filesavebox* when one wants the name of a file to save For example T=integer
M="*string*"	where the *string* is the text to in the box For example M="Enter a number"
O="*filename*"	where the *filename* is the name of a file in which to place the results for subsequent inclusion into GAMS For example O="file.inc"
D="*string 1*|*string 2...*"	where the "*string 1*|*string 2*|*string 3*|...|*string n*" gives the n strings to be associated with multiple choices when using *checkbox, radiobutton, combobox, or checklistbox*. The individual strings are separated by the delimiter "|" For example D="Small data set|Medium data set|Large data set"
E="*number 1*| *number 2...*"	where the "*number 1*|*number 2*|*number 3*|...|*number n*" gives the n numbers to be returned to GAMS associated with the choices made when using *checkbox, radiobutton, combobox, or checklistbox*. The individual numbers are separated by the delimiter "|" For example E="1|2|3|4|5"
I="*filepath*"	where *filepath* gives the path in which to look for the file under the

© 2022 Prof. Bruce McCarl

	fileopenbox and *filesavebox* dialogues. If not specified this is the project directory For example I="C:\gams\mine"
F="*filemask*"	where *filemask* gives the mask for acceptable files under the *fileopenbox* and *filesavebox* dialogues. If not specified this is *.*. For example I="*.gdx"
R="*string*"	where the *string* gives a line of GAMS code to place in the include file. This can contain a %s parameter in which the information to return is substituted For example R="$include '%s'" or R="set i /1990*%s/;"
C="*string*"	A title for the dialogue box being used For example C="Box to ask for a file"
L=*number*	where the *number* gives a lower bound on a numeric entry For example L=15
U=*number*	where the *number* gives an upper bound on a numeric entry For example U=15
@"filename"	where *filename* gives the name of a file of input instructions containing the options above in this table For example @ask.opt

In addition a number by itself can be entered to put multiple entries into columns under the *checkbox*, *radiobutton*, *combobox*, *or checklistbox* entries.

One can use GAMS to generate the input instruction file , but note that it is not possible to do this easily with the PUT facility since $call to ask is handled at compile time, before the PUT statement has done its work. Rather one must use $onecho and $offecho as follows

```
$onecho > asktest.opt
T=checklistbox
M=Choose multiple options
D=option 1|option 2|option 3|option 4|option 5
E=1|2|3|4|5
R=%s checked list box choice
O=k2.inc
$offecho
```

Then one would use the file as follows

```
$call =ask @asktest.opt
set k2 /
$include k2.inc
/;
display k2;
```

These options are discussed in ask and illustrated in ask.gms which contains the examples in the

© 2022 Prof. Bruce McCarl

documentation referred to above.

Example (ask.gms)

```
$call =ask T=integer M="Enter number of cities" o=n.inc
scalar n 'number of cities' /
$include n.inc
/;
display n;
```

More examples are in ask.gms and in the GAMS Data Utilities Models choice under model libraries in the IDE.

18.4.2 Msappavail

MSAPPAVAIL checks which Microsoft Office programs are installed.

The sequence

```
$call msappavail -option
```

can be used in GAMS to checks for the presence of the Microsoft Office software package option on the machine.

Allowable values of option:

```
-? List all known Office applications and their status
-Access
-Excel
-Explorer
-FrontPage
-Outlook
-PowerPoint
-Project
-Word
```

Upon return if called with an option other than -? then the GAMS recognized errorlevel is

set to 1 if the program looked for was not found.

set to 0 if found

Whether or not the item was found of not using

```
$if errorlevel 1 $goto noExcel
```

which is true if the item is not found. The file CTA.gms from the model library uses this feature.

© 2022 Prof. Bruce McCarl

18.4.3 Shellexecute

In some cases we want to let the computer figure out what application to start for a given document. This can be accomplished with ShellExecute written by Erwin Kalvelagen. For instance, when we call:
h

shellexecute demo.html

Windows will launch the web browser and show demo.html. This works correctly, irrelevant whether the user installed Microsoft Internet Explorer or Netscape's web browser.

Usage

The command line for ShellExecute looks like:

SHELLEXECUTE filename args

Additional parameters can be specified as documented in shellexecute They specifiy the action to be performed. ; how the application is to be displayed when it is opened; and the default directory for the sub-process.

In many cases you will not need to use these options.

18.4.4 Xlstalk

XLSTalk Allows communication with Excel to have it close files, see what version is running and do other functions.

Usage:

xlstalk <option> {-V} <file name> {<other parameters>}

where the parameters are

<file> : Optional Name of the Excel file with the file extension

<option> : One of the following options; see following table:

Option	Action	Parameter	Return code
-A	Test if Excel is running		0 if Excel not running, 1 if Excel is running
-C	Close file; do not save changes.	<file name>	

© 2022 Prof. Bruce McCarl

-E	Test if file exists	<file name>	0 if File does not exist, 1 if File exists
-M	Status of file	<file name>	0 if file is Not open in Excel 1 if File is open and is not modified 2 if File is open and has been modified
-O	Open file but do not reload	<file>	
-Q	Quit Excel if no workbooks have been modified	<file>	0 if Excel is closed, 1 if Excel is still running
-R	Run a macro in Excel	<file> <macro-name> {<macro-param>}	
-S	Save and close the file	<file>	
-V	Verbose mode		
-W	Wait for the user to close the file	<file>	
-X	Excel version		0 Excel is not installed 1 Excel is installed; version 2003 or earlier 2 Excel is installed; version 2007 or later

In GAMS, the return value can be obtained by using 'errorlevel'.

```
execute 'xlstalk.exe -X';
scalar x;
x = errorlevel;
display x;
```

18.5 Zip Utilities

Versions of zip and unzip are included in the GAMS releases and are called gmszip and gmsunzip.

These programs are located in the GAMS system directory.

To zip files one uses the execute or $call syntax plus the command

```
     gmszip archivename [-options] [-b path] [-t mmddyyyy]
[-n suffixes] [zipfile list] [-xi list]
```

as documented at http://infozip.sourceforge.net/Zip.html (note you need to add the gms prefix to use the version released with GAMS)

or in its simplest form

© 2022 Prof. Bruce McCarl

```
gmszip archivename file1 file2 ... filen
```

which will place file through filen in the archive named `archivename.zip`

To unzip files one uses the execute or $call syntax plus the command

```
gmsunzip file[.zip] [-Z] [-opts[modifiers]]  [list] [-x xlist] [-d exdir]
```

as documented at http://infozip.sourceforge.net/UnZip.html (note you need to add the gms prefix to use the version released with GAMS)

or in its simplest form

```
gmsunzip archivename -d c:\tmp\contentgoes_here
```

which will place all files in the archive named `archivename.zip` in the directory `c:\tmp\contentgoes_here`

19 Solver Option Files

Most of the GAMS solvers allow users to affect aspects of their operation by specifying an options file. In general, users do not need to exercise such options, as the default settings are usually appropriate for most problems. However, in some cases, it is possible to improve solver performance by specifying non-standard values. This chapter shows how to use option files.

Basics
Option file contents
Writing options during a model run
Transitory nature of options

19.1 Basics

Telling a solver to look for an options file: .Optfile
Option file name

19.1.1 Telling a solver to look for an options file: .Optfile

If you want to cause a solver to use an option file, it is necessary to set the optfile model attribute to a positive value. For example,

```
model mymodel /all/ ;
mymodel.optfile = 1 ;
solve mymodel using NLP maximizing dollars ;
```

© 2022 Prof. Bruce McCarl

19.1.2 Option file name

You must also create a file that contains the options to be passed on to the solver. The name of that file is specific to the solver being used. In particular the file name is

```
solvername.ext,
```

where solvername is the name of the solver that is currently being used to solve this problem type and is either the default solver or the one defined in a statement like:

```
option modeltype=solvername;
```

The file extension ext depends on the value to which the optfile model attribute has been set. If the model contains a statement like

```
mymodel.optfile = 1 ;
```

the file extension is opt. Setting this to values other than one changes this extension name in a manner as covered below.

For example, if we are solving a

- NLP and the active NLP solver is CONOPT the option file would be called conopt.opt;
- LP and the active LP solver is CPLEX the option file would be called cplex.opt.
- MINLP active MINLP solver is DICOPT the option file would be called dicopt.opt.

19.1.2.1 Alternative option file extention names: .Opt, .Op?, .O??, .???

By setting the value of optfile to a number different from 1 then one can have multiple, situation dependent, option files for the same solver. The following alternatives apply.

Modelname.optfile setting	Resultant File Extension		Example Name
0	--	(no option file)	--
1	.opt		Minos.opt
2	.op2		Conopt.op2
3	.op3		Cplex.op3
…			
10	.o10		Xa.010
…			
99	.o99		Dicopt.099
100	.100		Gamschk.100
…			
999	.999		Gurobi.999

© 2022 Prof. Bruce McCarl

In general

- no option file is assumed unless the optfile attribute is set to a non zero value
- if the option value is greater than or equal to 2 it is masked on top of the opt file extension with 999 being the largest value allowed.
- Using such names can be convenient so one can have different option file settings for different cases.

19.2 Option file contents

Option file contents vary from solver to solver. This section illustrates some of the common features of the file format. Users should refer to solver manual to discover the solver specific potential contents of an option file.

The option file is an ASCII text file containing one or more lines. Each line of the file falls into one of two categories

- A comment line
- An option specification line

Comments: *
Option specifications

19.2.1 Comments: *

A comment line may be inserted into an options file. Such a line begins with an asterisk (*) in the first column, and is not interpreted by either GAMS or the solver, and is used purely for documentation. One can also temporarily deactivate options using such an entry.

19.2.2 Option specifications

Each option specification line can contain only one option. The format for specifying options is as follows,

```
optionkeyword1 modifier or value
optionkeyword2 modifier or value
optionkeyword3 modifier or value
...
```

Notes:

- The option keyword may consist of one or more words and is not case sensitive.
- Modifiers are generally text strings but are not always required
- Numerical values are not always required and when entered may be either an integer or a real constant. Real numbers may be expressed in F, E, or D formats.
- Any errors in the spelling of the keyword(s) or modifiers will lead that option to not be understood

© 2022 Prof. Bruce McCarl

and therefore disregarded by the solver.

Examples:

Consider the following CPLEX options file,

```
* CPLEX options file
barrier
IIS yes
preind 0
```

The first line begins with an asterisk and therefore contains comments. The barrier entry specifies the use of the barrier algorithm to solver the linear programming problem, while the IIS indicates the irreducible infeasible set option is to be used on infeasible models and preind turns off the presolve. Details on these options can be found in the **CPLEX solver manual** .

Also consider the following MINOS options file,

```
Major iterations 2000
Scale all variables
feasibility tolerance 1.0e-5
```

The first option sets the major iteration limit to 2000. The second line tells MINOS to scale all variables including the ones with nonlinear terms. The third option sets the feasibility tolerance. Details on these options can be found in the **MINOS solver manual** .

19.3 Option file editor

The IDE contains an option file editor that can be used to

- Create a new options file for a solver
- Edit an existing option file.
- Look up option file possibilities and receive information on settings

The option file editor is invoked through the IDE **utilities>option editor** menu choice.

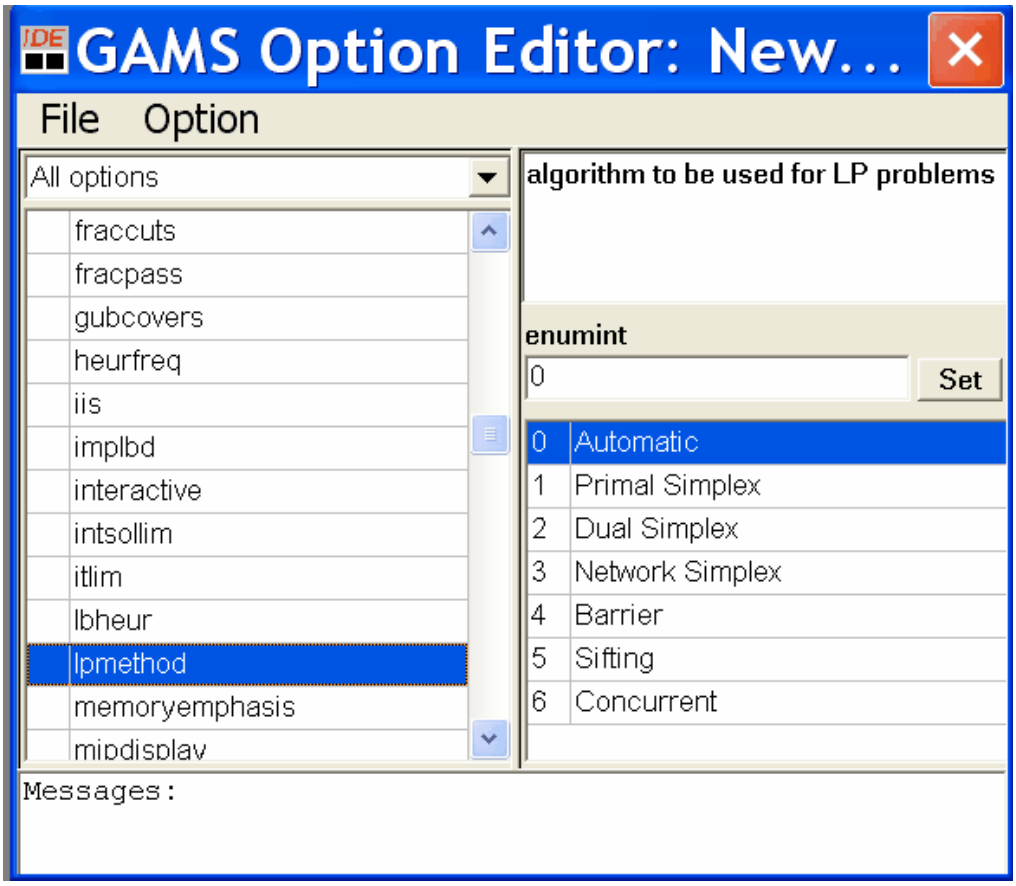
Once in the editor one can use the **file** menu to

- Read an existing option file
- Start up a new one for a particular solver or for GAMS although this seems to have no effect on GAMS execution)
- Save the file

Once an option file has been read or a new option file started with a solver chosen the user gets

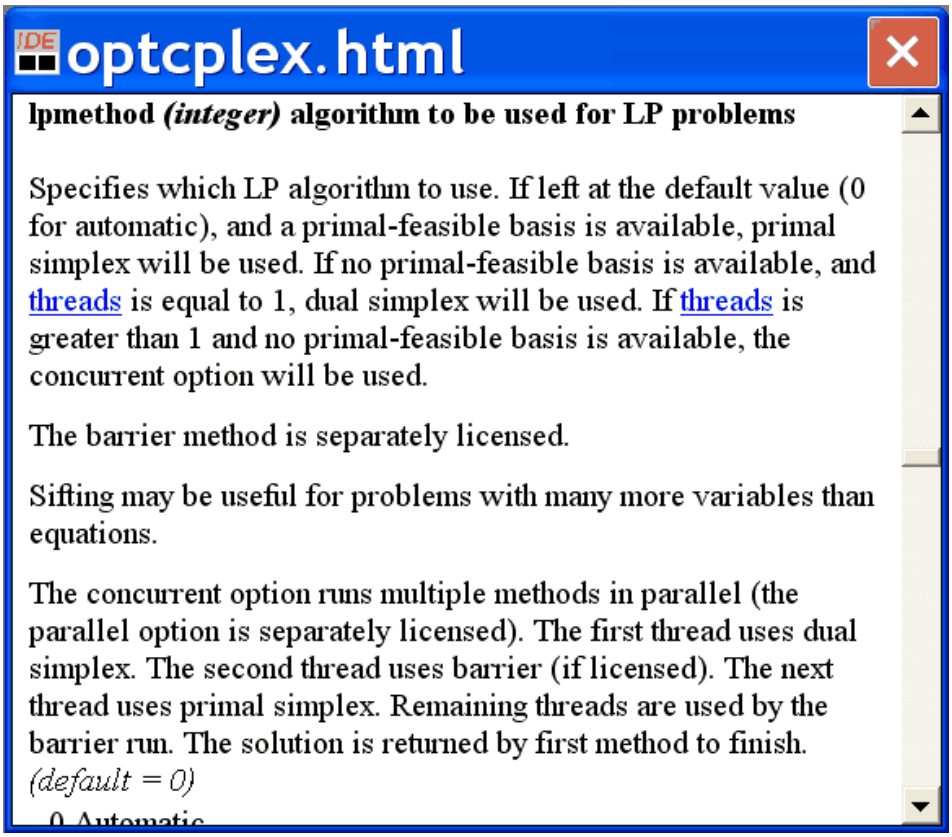
- A complete list of all alternatives that can be in the option file for the solver chosen
- A definition of the currently active option and a list of the alternative choices for that option. For example the following screen comes from a new option file after CPLEX was the chosen solver and the option LPMETHOD was scrolled down to

© 2022 Prof. Bruce McCarl



- One can also do a right mouse click or press F2 and get a further definition of the select option as below and in fact this opens a document of all options for the solver at hand.

© 2022 Prof. Bruce McCarl



- There is no attempt in the option file editor to group the options by function and users should still go to the solver guides for that material. This is acccessed through the IDE help menu.

19.4 Writing options during a model run

Users may wish to construct the options file during a job to insure that the option file contents are always current. This can be done using Put command as discussed in the Output via Put Commands chapter. In that case, the file must be closed prior to the SOLVE statement using a PUTCLOSE. The following example shows the creation and closing of an option file for the MINOS solver as implemented in **frstpart.gms**.

```
FILE  OPT   MINOS option file  / MINOS5.OPT /;
PUT OPT;
PUT 'BEGIN'/
    '  Iteration limit        500'/
    '  Major damping parameter 0.5'/
    '  Feasibility tolerance   1.0E-7'/
    '  Scale all variables'/
    'END';
PUTCLOSE OPT;
```

© 2022 Prof. Bruce McCarl

Learning about options: Solver manuals
Default settings for Optfile
Defining a central location for the option files: Optdir

19.5 Learning about options: Solver manuals

Each solver manual explains the specific available solver options. They can be obtained off the distribution disk, in the GAMS systems directory DOCS subdirectory(c:\program files\gams22.7\docs\ today on a Windows machine) or through the IDE.

19.6 Default settings for Optfile

One can preset a value for the optfile command line parameter in the GMSPRMXX.txt customization file for the computer to provide a default value for optfile of 1 automatically making the solver always look for the opt file if desired. Procedures for setting such parameters appear in the Customizing GAMS chapter.

19.7 Defining a central location for the option files: Optdir

One can specify a default location where GAMS will look for option files using the command line parameter Optdir. If not specified, it will be set to the current working directory. A customization value of Optdir may also be set in the Gmsprmxx.txt customization file.

19.8 Transitory nature of options

Solvers are continually updated adding new features. This occasionally makes new options available or changes the format of old ones as solvers are improved. The solver manual is always the best current reference.

20 Advanced Basis Usage

A substantial amount of the solution time encountered when solving a mathematical programming model involves finding the appropriate variables to be in the solution. Using common mathematical programming terminology – this involves a search for the optimal Basis. Often solution time can be reduced substantially if one can identify what to include a priori by suggesting what is known as an **Advanced Basis**. Here I discuss how to do that in GAMS. Note that these techniques do not work with MIPs or with the non-simplex based algorithms like Barrier in CPLEX. However, they do work quite well in many LP, NLP, and MCP applications.

Basics
Advanced basis formation in GAMS
Effect of advanced basis on solution performance
Bratio
Providing a basis
Guessing at a basis

© 2022 Prof. Bruce McCarl

Problems with a basis

20.1 Basics

Consider the benefits of a basis in a linear programming context. The solution to a linear programming problem generally has one non-zero variable for each constraint. This means in a model with M constraints all one really needs to know is exactly which M variables are non-zero.

Generally, linear programming solvers take three or more times as many iterations as the number of constraints to reach a solution. Simplex based linear programming solvers initially guess which M variables should be in the solution then one by one insert better variables into the basis until the ultimate, optimal basis is found.

If one can supply an improved guess specifying the set of variables to be in the basis (those to be non-zero in the final optimal solution) then solution time can be reduced. An advanced basis is a user-defined suggestion on variables in the solution. A good advanced basis can reduce solution time by more than an order of magnitude. In one case within my work advanced basis suggestions caused a reduction in solution time from 36 hours to 1 hour.

Here I cover how to provide such information and also discuss some of the difficulties that may arise when using advanced bases.

20.2 Advanced basis formation in GAMS

In GAMS an advanced basis is formed whenever sufficient information is available on the levels and marginals for basis formation. The exact information used are:

- Variables with non-zero levels are suggested to be basic unless the levels fall at the upper or lower bounds.
- Variables with non-zero levels equal to their upper or lower bounds – (x.l=x.up or x.l=x.lo) are not suggested to be basic rather being suggested as held at their bound. Variables held at bound should also have non-zero marginals (x.m).
- Variables with zero levels and non-zero marginals are not suggested to be in the basis.
- Constraints with zero marginals have their slacks suggested for inclusion in the basis.
- Constraints with non-zero marginals are treated as binding constraints.

Such information is typically not available the first time a model is solved unless the user has explicitly provided it.

20.3 Effect of advanced basis on solution performance

The effect of an advanced basis on solver performance is best illustrated by example. Suppose I take a model and solve it twice (**twotran.gms**) and observe what happens in comparison with solving the model alone. In particular, I solve a model first then I solve it again then I alter the objective function and solve it a third time. The code to do this appears below (**twotran.gms**).

```
SOLVE FIRM USING LP MAXIMIZING NETINCOME;
SOLVE FIRM USING LP MAXIMIZING NETINCOME;
TRANSCOST(PRODUCT,TYPE,PLANT,PLANTS)
```

© 2022 Prof. Bruce McCarl

```
     =TRANSCOST(PRODUCT,TYPE,PLANT,PLANTS)*3;
SOLVE  FIRM USING LP MAXIMIZING NETINCOME;
```

The first solve of the model takes 11 iterations the second one takes 0 and the third one takes 1. The model in the third solve when solved all by itself (**twotrana.gms**) takes 12 iterations.

The reason for fewer iterations when the second and third solves are issued is that GAMS suggested an advanced basis using the saved solution information from the prior solve.

20.4 Bratio

Generation of a basis is actually controlled by the setting of Bratio.

GAMS will suggest a basis as long as the number of candidates for basis inclusion exceed the number of needed elements times the value of Bratio. Thus in a 1000 equation LP with the default value of Bratio (0.25) GAMS will not suggest a basis unless it could find at least 250 elements to include in it. Generally this would not be the case unless the problem had been previously solved and then all 1000 elements would be known or other efforts are made. This means a basis is always suggested by default as long as I are dealing with a second or later solve. It also means that by setting Bratio to 1.0 one can cause the basis to never be suggested and by setting it to a small number (0.01) one can cause GAMS to use whatever is available.

20.5 Providing a basis

So if you want to suggest a basis how can you do it? This happens under three conditions

- Repeatedly solving
- Using a point GDX file
- Guessing at levels and marginals

each of which is explained below.

Getting a basis through repeated solution
An alternative – use a GDX point file

20.5.1 Getting a basis through repeated solution

GAMS always tries to suggest an advanced basis using the saved solution information from a previous solve. But if a solve has not been done then a basis cannot be provided. So users wishing to get whatever gains they can should to the extent possible try to retain a base model solution then execute subsequent solves.

20.5.1.1 Save files

Often one can retain a base model solution through the use of saved work files and in turn provide a starting basis for restarted GAMS code modules that contain subsequent solves. An example of this is inherent in the following DOS bat file (**seq.bat**).

```
GAMS frstpart s=f1
GAMS nextpart r=f1
```

© 2022 Prof. Bruce McCarl

where both parts (**frstpart.gms**, **nextpart.gms**) have solve statements in them.

In turn, once a modeler is satisfied with the model inherent in the saved work files one can retain that work file set providing a basis to all restarted GAMS code. Then one can set up alternative runs modifying model data in files that will be subsequently restarted from those work files. The Achilles heel in this case is that sometimes the original data or model structure needs to be modified and one must work within the original file without the possibility of having GAMS automatically retaining the information from a previous solve.

20.5.2 An alternative – use a GDX point file

For many years applied linear programmers have developed advanced bases by causing a solver to write out a basis and then load that basis when doing the solution of a related model. This is done under the assumption that such a solution will be close to the solution for the related model. GAMS can automatically save and load a file with such information in the form of a GDX point file through the employment of the Savepoint parameter and the subsequent loading of the GDX file using Execute_loadpoint. The steps to use the GDX point file are:

I. Run a model from which you wish to save the basis information using the Savepoint option, model attribute, or command line parameter.

   a. Insert the command

```
option Savepoint=1;
```

   or

```
option Savepoint=2;
```

   before the solve statement.

   This will cause a file to be saved which contains the model solution information. The file name for savepoint=1 will be your model name plus the suffix _p.gdx. Thus, if your model is named transport then a file named transport_p.gdx. will be saved and if the model is named farmmod then the basis file will be named farmmod_p.gdx. In the case of multiple solves, the basis from the last solve after the command will be the one that is saved. When savepoint=2 is used files are saved for every solve as discussed in the GDX, option, model attribute, or command line parameter chapters.

II. Incorporate the basis in any file in which you wish the basis to be used. In particular use the Execute_loadpoint syntax to incorporate the saved basis file somewhere after the variable and equation definitions but before the solve statement

```
Execute_loadpoint 'transport_p.gdx';
```

Example:

(makepointbas.gms)

First to solve the base problem and save the basis the statements from the model on are

```
MODEL FIRM /ALL/;

OPTION Savepoint=1;;
```

© 2022 Prof. Bruce McCarl

```
solve firm using LP maximizing objfun;
```

This solution takes two iterations. The resultant GDX point file contains the marginals and levels for the variables and equations.

I may now modify my base model to include and Execute_loadpoint command that loads the basis (**loadpointbas.gms**). I may or may not continue use of the Savepoint option

```
MODEL FIRM /ALL/;
OPTION SOLPRINT = ON ;
* SECTION D    SOLVE THE PROBLEM
execute_loadpoint 'firm_p';
SOLVE FIRM USING LP MAXIMIZING NETINCOME;
```

The resultant solution takes zero iterations. Thus given the basis, I did no work at all.

20.5.3 GAMSBAS

GAMSBAS was an older procedure to write a basis. It was eliminated some time ago but its functionality can be duplicated with use of GDXDUMP.

20.6 Guessing at a basis

One can try to guess at a solution by specifying

- Non-zero marginals for the constraints that are felt to be binding.
- Non-zero levels for the variables that are felt to be non-zero in the solution as follows.
- Non-zero marginals for variable not felt to be in the solution.

For example,

```
OBJECTIVE.m        =    1 ;
RESOURCE.m ("R1")  =    1 ;
RESOURCE.m ("R2")  =    1 ;
RESOURCE.m ("R3")  =    0 ;
OBJFUN.l           =    1 ;
X.l ("X1")         =    1 ;
X.l ("X2")         =    1 ;
X.m ("X3")         =    1 ;
```

This basis if included would cause the simple problem to solve in zero iterations. But making this kind of guess for a large model would be hard. Guess in the NLP case must be done with care as the variable levels are more important and provide the starting point for evaluating the Jacobian.

20.7 Problems with a basis

Provision of an advanced basis does not always help. Solvers may perform poorly encountering problems because the basis is poor. In addition, certain problem types are not amenable to advanced basis use (MIPs). Presolves may also cause the basis to be ignored.

© 2022 Prof. Bruce McCarl

Symptoms and causes of a poor advanced basis
MIP
NLP

20.7.1 Symptoms and causes of a poor advanced basis

Sometimes a basis may not help and one may find the solver fails or uses excessive iterations. This can be particularly true in a repeated set of solutions with a radically altered model. For example, one can observe messages after a solve such as

```
sorry guys we seem to be stuck or
after 3 factorizations the basis is singular
```

or the solver may also not make significant progress.

There are a couple of possible underlying causes of such messages

- One may have altered the model size using conditionals which either:
  - Eliminated a large number of variables in the basis.
  - Deleted a large number of constraints.
- One may also have caused a radical alteration in the model coefficients that compromised the basis.
- One may be using a basis from a model that was radically altered that possesses many features that the model on hand does not have.

There are several ways of either avoiding the problems such a basis causes or insuring that an advanced basis would be more compatible between related models

- To increase compatibility don't change model size, eliminate things economically as illustrated in the Doing a Comparative Analysis with GAMS chapter.
  - Rather than eliminate a variable leave it in the model but alter it's objective function coefficient so it has a very high cost and will not be desirable to include in the solution.
  - Rather than eliminate a constraint make it non-binding by for example in the case of =L= constraints adding a very large constant to the right hand side.
- Revert to a saved basis from a related but less altered model.
- Tell the solver to dump the basis using the BRATIO=1 option or the model attribute myModel.`defPoint`=1.

20.7.2 MIP

Mixed integer programming problem bases may not be all that successful as what's really needed is storage of the branch and bound tree. The only real success from bases may be giving an initial good objective function bound and getting the linear programming part of the solver started relatively quickly.

© 2022 Prof. Bruce McCarl

20.7.3 NLP

In nonlinear models advanced bases are often helpful, but there are cases where they do not contribute to shortened solution time. There are no general rules. Only experience with a problem will show the effectiveness of the advanced basis. Nonlinear programming is the place where I have encountered the biggest solution time reductions.

21 Mixed Integer, Semi, and SOS Programming

There are a number of features within GAMS that are designed for use in formulating and solving mixed integer, semi integer, semi continuous and specially ordered set (SOS) programming problems all of which require a mixed integer or MIP solver.

Specifying types of variables
Imposing priorities
GAMS options and model attributes
Branch and bound output
Nonlinear MIPs
Identifying the solver
Model termination conditions and actions
Things to watch out for
Branch and Cut and Heuristic Facility

21.1 Specifying types of variables

The following types of variables fall into the mixed integer programming category in GAMS

- Binary variables (Binary). These can only take on values of 0 or 1.
- Integer variables (Integer). These can take on integer values between a lower and an upper bound. By default these variables are bounded in GAMS to the interval 0 to 100.
- Specially Ordered Sets Type 1 (SOS1). Groups of variables where only one member in each group can have a nonzero value in the solution.
- Specially Ordered Sets Type 2 (SOS2). Groups of variables where only two variables in the group can have nonzero solution levels and they must be adjacent.
- Semi-continuous variables (Semicont). Variables that must either be zero or can take on a continuous value above a threshold value.
- Semi-integer variables (Semiint). Variables that must be must either be zero or can take on a integer value above a threshold limit.

Each is discussed below.

Binary variables
Integer variables
Specially ordered set variables of type 1 (SOS1)
Specially ordered set variables of type 2 (SOS2)

© 2022 Prof. Bruce McCarl

Semi-continuous variables
Semi-integer variables

21.1.1 Binary variables

These can take on values of 0 or 1 only. Binary variables are declared as follows

```
Binary Variable s1(i), t1(k,j), w1(i,k,j) ;
```

Example:

(basint.gms)

```
POSITIVE VARIABLE       X1
INTEGER VARIABLE        X2
BINARY VARIABLE         X3
VARIABLE                OBJ
EQUATIONS               OBJF
                        X1X2
                        X1X3;
OBJF..     7*X1-3*X2-10*X3 =E= OBJ;
X1X2..     X1-2*X2 =L=0;
X1X3..     X1-20*X3 =L=0;
option optcr=0.01;
MODEL IPTEST /ALL/;
SOLVE IPTEST USING MIP MAXIMIZING OBJ;
```

Notes:

- The lower bound of zero and upper bound of one restrictions do not need to be added as they are automatically generated.
- Often such variables are used in generating logical conditions such as imposing mutual exclusivity, complementarity, or other types of phenomenon as discussed in *Chapter 15 of McCarl and Spreen*.
- Priorities (.prior attributes of variables) can be used to override binary specifications as discussed below.

21.1.2 Integer variables

These can take on integer values between specified lower and upper bounds where note the default upper bound is 100. Integer variables are declared as follows,

```
Integer Variable s1(i), t1(k,j), w1(i,k,j) ;
```

Example:

(**basint.gms**)

```
POSITIVE VARIABLE       X1
INTEGER VARIABLE        X2
BINARY VARIABLE         X3
```

© 2022 Prof. Bruce McCarl

```
VARIABLE                OBJ
EQUATIONS               OBJF
                        X1X2
                        X1X3;
OBJF..      7*X1-3*X2-10*X3 =E= OBJ;
X1X2..      X1-2*X2 =L=0;
X1X3..      X1-20*X3 =L=0;
x2.up=125;
option optcr=0.01;
MODEL IPTEST /ALL/;
SOLVE IPTEST USING MIP MAXIMIZING OBJ;
```

Notes:

- These variables are automatically bounded by GAMS so they have a default upper bound of 100. If the user wishes the integer variables to take on values greater than 100, a larger bound must be specified.
- A lower bound of zero is automatically generated. This may also be changed.
- Priorities (.prior attributes of variables) can be used to override binary specifications as discussed below.

21.1.3 Specially ordered set variables of type 1 (SOS1)

At most one variable within a specially ordered set of type 1 (SOS1) can have a non-zero value. This variable can take any positive value. SOS1 variables are declared as follows:

```
SOS1 Variable s1(i), t1(k,j), w1(i,k,j) ;and
```

The members of the right-most index for each named item are defined as belonging to the SOS1 group or set of variables of which at most one of which can be non zero.

For example, in the SOS1 variables defined above,

- s1 forms one group of mutually exclusive SOS1 variables which contains elements for each member of the set i and thus only one variable for one of the cases of i can be nonzero with the rest being zero.
- t1 defines a separate SOS1 set for each element of k and within each of those sets the variables indexed by j are SOS1 or mutually exclusive.
- w1 a separate SOS1 set for each pair of elements in i and k and within each of those sets the variables indexed by j are SOS1 or mutually exclusive.

Example:

prodschx.gms from the GAMS model library shows formulations with binary, SOS1 and SOS2 sets.

Notes:

© 2022 Prof. Bruce McCarl

- By default each SOS1 variable can range from 0 to infinity. As with any other variable, the user may set these bounds to whatever is required.
- One is required to utilize a mixed integer (MIP) solver to solve any model containing SOS1 variables. However, the SOS1 variables do not have to take on integer solution levels.
- The MIP solver is required because the solution process needs to impose mutual exclusivity and to do this it implicitly defines an additional set of zero one integer variables, then solves the problem as a MIP.
- The user can provide additional constraints say requiring the sum to the SOS1 variables in a set to be less than or equal to a quantity (often 1 for convexity). Consider the following example,

```
SOS1 Variable s1(i) ;
Equation defsoss1 ;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;
```

  Here the equation defsoss1 defines the largest non-zero value 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 has to be nonzero and equal to a number. In this case, the defSoss1 equation will be

```
defSoss1.. sum(i,s1(i)) =e= 10 ;
```

  A common use of the use of this type of restriction is for the case where the right hand side in the equation above is 1. In such cases, the SOS1 variable is effectively a binary variable. In such a case, the SOS1 variable could just have been binary and the solution provided by the solver would be indistinguishable from the SOS1 case.

- Not all MIP solvers allow SOS1 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. A model that contains these variables may not be perfectly transferable among solvers. You should verify how the solver you are using handles SOS1 variables by checking the relevant section of the solver manual.

21.1.4 Specially ordered set variables of type 2 (SOS2)

At most two variables within a specially ordered set of type 2 (SOS2) can take on non-zero values. The two non-zero values have to be for adjacent variables in that set. Specially ordered sets of type 2 variables are declared as follows:

```
SOS2 Variable s2(i), t2(k,j), w2(i,k,j) ;
```

The members of the right-most index for each named item are defined as belonging to a special (SOS2) group or set of variables of which at most one of which can be non zero.

For example, in the SOS2 variables defined above,

- s2 forms one group of SOS2 variables of which at most 2 can be non zero and they must be adjacent in terms of the set i. The adjacency means if the set i has elements /a,b,c,d,f,g/ that one could have any 2 variables like the ones associated with set elements a and b but never a and c since the set elements are not adjacent. This means the sets used must be ordered as discussed in the Sets chapter.
- t2 defines a separate SOS2 set for each element of k and within each of those sets no more

© 2022 Prof. Bruce McCarl

than 2 variables can be non zero. Further, they they must be adjacent in terms of the set j. The adjacency means if the set j has elements /j1,j2,j3,j4,j5,j6/ that one could have any 2 variables like j3 and j4 but never j1 and j6 since the set elements are not adjacent. This means the set j must be ordered as discussed in the Sets chapter.

- w2 defines a separate SOS2 set for each pair of elements in i and k. Within each of those sets no more than 2 variables can be non zero and they must be adjacent in terms of the set j. The adjacency means if the set j has elements /j1,j2,j3,j4,j5,j6/ that one could have any 2 variables like j3 and j4 but never j2 and j4 since the set elements are not adjacent. This means the set j must be ordered as discussed in the Sets chapter.

Example:

prodschx.gms from the model library shows formulations with binary, SOS1 and SOS2 sets.

Notes:

- The most common use of SOS2 sets is to model piece-wise linear approximations to nonlinear functions using separable programming.
- One must use a mixed integer (MIP) solver to solve any model containing SOS2 variables. But, the SOS2 variables do not have to take on integer solution levels.
- The MIP solver is required because the solution process needs to impose both adjacency restrictions and the restrictions that no more than 2 nonzero level values can be present and to do this the solvers implicitly defines an additional set of zero one variables, then solves the problem as a MIP.
- The default bounds for SOS2 variables are 0 to plus infinity. As with any other variable, the user may set these bounds to whatever is required.
- Not all MIP solvers allow SOS2 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Thus a model that contains these variables may not be perfectly transferable among solvers. Please verify how the solver you are using handles SOS2 variables by checking the relevant section of the Solver Manual.

21.1.5 Semi-continuous variables

Semi-continuous variables are restricted, if non-zero, to take on a level above a given minimum and below given maximum. This can be expressed algebraically as:

Either

```
x=0
```

or

```
x $ a and x ≤ b
```

By default, the lower bound (a) is 1.0 and the variable is upper bounded at infinity. The lower and upper bounds are set through the .lo and .up variable attributes as discussed in the Variables, Equations, Models and Solves chapter. In GAMS, a semi-continuous variable is declared using the reserved phrase Semicont variable. The following example illustrates its use.

```
semicont variable x ;
```

© 2022 Prof. Bruce McCarl

```
x.lo = 1.5 ; x.up = 23.1 ;
```

The above code declares the variable x to be a semi-continuous variable that can either be 0, or can behave as a continuous variable between 1.5 and 23.1.

Notes:

- One is required to utilize a mixed integer (MIP) solver to solve any model containing Semi-continuous variables. However, these variables do not have to take on integer solution levels.
- The MIP solver is required because the solution process needs to impose the discontinuous jump between zero and the threshold value. To do this solvers implicitly define an additional zero one variable, and then solve the problem as a MIP.
- 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.
- Not all MIP solvers allow semi-continuous variables. Please verify that the solver you are using can handle semi-continuous variables by checking the solver manual.

21.1.6 Semi-integer variables

Semi-integer variables are restricted, if non-zero, to take on an integer level above a given minimum level. This can be expressed algebraically as:

Either

```
x=0
```

or

```
x $ a and integer
```

By default, the lower bound (a) is set to 1.0 and the variable is upper bounded at 100. The lower and upper bounds are set through the .lo and .up variable attributes as discussed in the Variables, Equations, Models and Solves chapter.

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 = 23 ;
```

The above declares the variable x to be a semi-continuous variable that can either be 0, or can behave as an integer variable between 2 and 23.

Notes:

- One is required to utilize a mixed integer (MIP) solver to solve this problem type.
- 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.

© 2022 Prof. Bruce McCarl

- The variables are upper bounded at 100. If one wants larger bounds then they need to be specified.
- The bounds for semiint variables have to be set at integer values. GAMS will flag an error during model generation if it finds that this is not the case.
- 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 Solver Manual.

21.2 Imposing priorities

In MIP models users can specify an order for picking variables to branch on during a branch and bound search. This is done through the use of priorities. Without priorities, the MIP algorithm will internally determine which variable is the most suitable to branch on.

Priorities are set for individual variables through the use of the .prior variable attribute as discussed in the Variables, Equations, Models, and Solves chapter. As a general rule of thumb, the most important variables should be given the highest priority which implies they should have the lowest nonzero values of the .prior attribute. The default value is 1.0. Functionally **.prior** establishes in what order variables are to be fixed to integral values while searching for a solution. Variables with a specific .prior value will remain relaxed until all variables with lower .prior values have been fixed. A .prior value of +Inf indicates the variable will always be relaxed, ie not discrete.

For priorities other than the infinity one to be used the user must activate them through use of the GAMS model attribute statement

```
mymodel.prioropt = 1 ;
```

where mymodel is the name of the model specified in the model statement for the problem to be solved as discussed in the Model Attributes chapter. The default value is 0 in which case priorities will not be used.

Example:

The following example illustrates its use,

```
mymodel.prioropt = 1 ;
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.

Notes:

- The higher the value given to the .prior suffix, the lower the priority for branching.
- Note that there is a prior variable attribute for each individual component of a multidimensional variable.
- All members of any SOS1 or SOS2 set should be given the same priority value.

© 2022 Prof. Bruce McCarl

- The **.prior** attribute of a discrete variable can be used to relax the discrete restriction on that variable. Setting the .prior value to +inf will relax a variable permanently. This relaxation is done independent of the model attribute .prioropt.

21.3 Branch-and-Cut-and-Heuristic Facility

Hard mixed-integer programming (MIP) problems can be solved faster with the help of user supplied routines that generate *cutting planes* and good *integer feasible solutions*.The GAMS Branch-and-Cut-and-Heuristic (BCH) facility closes this gap. It automates all major steps necessary to define, execute and control the use of user defined routines within the framework of general purpose MIP codes. It is documented at BCH Facility.

21.4 GAMS options and model attributes

GAMS has a number of options and model attributes that can be used to influence MIP solver performance or report on MIP results. They are invoked as discussed below or in the Model Attributes chapter.

Modelname.Cheat = x
Modelname.Cutoff = x
Modelname.Nodlim = x
x=Modelname.objest
Modelname.Optca=X Option Optca=X
Modelname.Optcr=X Option Optcr=X
Modelname.Optfile = 1
Modelname.Prioropt = 1
Modelname.Tryint = x

21.4.1 Modelname.Cheat = x

The cheat value requires each new integer solution to be at least x better than the previous one. This can reduce the number of nodes that the MIP solver examines and can improve problem solving efficiency. However, setting this option at a positive value (zero is the default) can cause some integer solutions, including the true integer optimum, to be missed. When a model has been solved with the cheat parameter set at a nonzero level than all one is able to say is that the optimum solution is within the cheat parameter or less of the solution found. The cheat parameter is specified in absolute terms (like the Optca option). Certain solver options override the cheat setting.

Use of this parameter is done using a command like (**basint.gms**)

```
iptest.cheat=0.1;
```

where the model being solved is named iptest and cheat is set to 0.1.

Integer programming solver option file parameters like the CPLEX option objdif can override the cheat

© 2022 Prof. Bruce McCarl

attribute value.

21.4.2 Modelname.Cutoff = x

As the branch and bound search proceeds, the parts of the tree with an objective worse than the cutoff value x are ignored. This can speed up the initial phase of the branch and bound algorithm (before the first integer solution is found). However, setting this option at a positive value (zero is the default) can cause some integer solutions, including the true integer optimum, to be missed if in a maximization, it's value is above the cutoff. In fact, if the If you set cutoff below the optimum, you get no solution -- not just some missed integer solutions. Cutoff may also cause one to miss finding an initial feasible integer solution.

The cutoff parameter is specified in absolute terms (like the Optca option).

Use of this parameter is done using a command like (**basint.gms**)

```
iptest.cutoff=12;
```

where the model being solved is named iptest and cutoff is set to 12.

21.4.3 Modelname.Nodlim = x

This attribute specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. This can stop solutions that are exhibiting "excessive" iterations and if the limit is reached causes the algorithm to terminate, without reaching optimality. The Nodlim parameter is specified as an integer.

Use of this parameter is done using a command like (**secur.gms**)

```
security.nodlim=10000;
```

where the model being solved is named security and nodlim is set to 10000.

21.4.4 x = Modelname.objest

Some MIP capable GAMS solvers generate a bound on the objective function value for the best possible solution when solving integer programs .

Users can access this bound by using the model attribute `objest`.

One can access this by setting a parameter equal to Modelname.objest

```
xx=mymip.objest;.
```

or (secur.gms)

© 2022 Prof. Bruce McCarl

```
scalar bestobj  solver est of best obj value;
bestobj=security.objest;
display bestobj;
```

21.4.5 Modelname.Optca=X Option Optca=X

This specifies the absolute optimality criterion for a MIP problem. In general, GAMS tells the solvers to stop trying to improve upon the integer solution and stop calling the solution close enough to optimal when

```
(|BP - BF|) < Optca,
```

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution.

This reduces solution time as the solver stops not looking for better solutions. However, setting this option at a positive value (zero is the default) can cause the true integer optimum to be missed if it's value is within Optca of the best solution on hand when the problem stops. The final solution could be the best, but is guaranteed only to be within the tolerance of the "true optimal".

Optca is specified in absolute terms relative to the objective value. Thus a value of 100 means the objective value will be within the 100 units of the true objective value.

Use of this parameter involves a command like (**basint.gms**)

```
iptest.optca=12;
```

or

```
Option optca=12;
```

where the model being solved is named iptest and optca is set to 12.

21.4.6 Modelname.Optcr=X Option Optcr=X

This specifies the relative optimality criterion for a MIP problem. In general GAMS tells the solvers to stop trying to improve upon the integer solution when

```
(|BP - BF|)/(|BP|) < Optcr ,
```

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. However some solvers, in particular CPLEX use slightly different definitions. The Optcr option is used in CPLEX to stop when (|BP - BF|)/(1.0e-10 + |BF|) < Optcr.

In turn the solver stops after finding a solution proven to be "close enough" (within the Optcr tolerance) to optimal. This reduces solution time as the solver stops not looking for better solutions. However, setting this option at a positive value (0.1 is the default) can cause the true integer optimum to be missed if it's value is within Optcr of the best solution on hand when the problem stops. The final solution could be the best but is guaranteed only to be within the tolerance of the "true optimal".

© 2022 Prof. Bruce McCarl

The Optcr parameter is specified in proportional terms relative to the objective value thus a value of 0.10 means the objective value will be within the 10% of the true objective value.

Use of this parameter is done using a command like (**basint.gms**)

```
iptest.optcr=0.012;
```

or

```
Option optcr=0.012;
```

where the model being solved is named iptest and optcr is set to 0.012 or 1.2%. The default value for Optcr is large being 0.10 or 10%.

21.4.7 Modelname.Optfile = 1

Instructs the mixed integer solver to read an options file as discussed in the Solver Option Files chapter. The name of the option file is solvername.opt (ie cplex.opt or xa.opt or osl.opt). Solver options allow one to manipulate the way solvers work affecting a number of solver functions including choice of the branch and bound tree handing strategies. The solver manuals cover the allowable options.

21.4.8 Modelname.Prioropt = 1

Instructs the mixed integer solver to use priority branching information passed by GAMS through the variable.prior attributes. If used priorities should reflect knowledge of the problem. Variables with higher priorities – lower values of the ,prior attribute -- will be branched upon before variables of lower priorities. This information indicates user specified preferred directions for the internal branch and bound tree search and can dramatically reduce the number of nodes searched.

Problem knowledge may indicate what could be considered first. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to be purchased within the factory so you would set the priority values lower for the build variables. By assigning a higher priority – lower value of prior- to the build/nobuild decision variable, you can force this logic into the tree search and speed up computation time by not exploring uninteresting portions of the tree.

Note priorities are not needed and the branch and bound codes used sophisticated branching criteria involving potential of the variable to affect the objective function value.

21.4.9 Modelname.Tryint = x

Causes the mixed integer solver to try to make use of the available initial integer solution. The exact form of implementation depends on the solver and can be in part controlled by solver settings or options. See the solver manuals for details.

21.5 Branch and bound output

When the model **secur.gms** was solved with an earlier version of CPLEX it yielded output like the following

© 2022 Prof. Bruce McCarl

```
             Nodes                                                Cuts/
        Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap
           0     0   2.2303e+007    24                2.2303e+007         0
         100    93   2.2303e+007     1                2.2303e+007        53
         200   193   2.2303e+007     1                2.2303e+007        53
    *    280+  266   2.2303e+007     0   2.2303e+007   2.2303e+007        53    0.00%
         300   270   2.2303e+007     1   2.2303e+007   2.2303e+007        53    0.00%
    *    390    65   2.2303e+007     0   2.2303e+007   2.2303e+007       104    0.00%
    Fixing integer variables, and solving final LP..
    MIP Solution  :     22303062.100023    (104 iterations, 391 nodes)
    Final LP      :     22303062.100023    (0 iterations)
    Best integer solution possible :    22303113.765793
    Absolute gap          :            51.6658
    Relative gap          :       2.31653e-006
```

This output will differ across solvers but generally contains the same types of information showing the branch and bound approach in action. Namely the columns by label are

- Node is number of branch and bound problems examined so far.
- Nodes left is number of problems created during the branching process that are yet to be examined.
- IInf tells number of integer variables with non-integer solution levels.
- Objective gives the current objective function value.
- Best node gives the current lower bound on the solution. Similarly the column
- Best integer gives the incumbent solution. Note the last solution in that column is not necessarily global best.
- Gap gives max percentage difference from theoretical optimum.

Here we see, no solution is found for a while (indicated by blank entry in Best Integer until iteration 280), then one found and another. This shows the common phenomena that MIP solves usually end with a gap between the solution found and the best possible. This is controlled by iteration limits, resource limits, solver options, and model attributes like optcr/optca.

21.6 Nonlinear MIPs

Modelers may wish to impose integer restrictions on nonlinear formulations. Today GAMS contains the DICOPT and CBC solvers that permit this. They tie together other solvers. For example both can use CONOPT to solve the nonlinear sub-problems. DICOPT also uses MIP solvers on the integer part of the problem while CBC contains an internal integer solution algorithm.

For example suppose we impose restrictions in a portfolio problem that a minimum of 10 shares be bought if any and that we buy integer numbers of shares (**intev.gms**)

```
Integer VARIABLES  INVEST(STOCKS)       MONEY INVESTED IN EACH STOCK
binary variables   mininv(stocks)       at least 10 shares bought
VARIABLE              OBJ               NUMBER TO BE MAXIMIZED ;
EQUATIONS             OBJJ              OBJECTIVE FUNCTION
```

© 2022 Prof. Bruce McCarl

```
                        INVESTAV          INVESTMENT FUNDS AVAILABLE
                        minstock(stocks)  at least 10 units to be bought
                        maxstock(stocks)  Set up indicator variable ;
OBJJ.. OBJ =E=   SUM(STOCKS, MEAN(STOCKS) * INVEST(STOCKS))
               - RAP*(SUM(STOCK, SUM(STOCKS,
                        INVEST(STOCK)* COVAR(STOCK,STOCKS) * INVEST(STOCKS))));
INVESTAV..     SUM(STOCKS, PRICES(STOCKS) * INVEST(STOCKS)) =L= FUNDS;
minstock(stocks)..    invest(stocks) =g= 10*mininv(stocks);
maxstock(stocks)..    invest(stocks)=l=1000*mininv(stocks);
MODEL EVPORTFOL /ALL/ ;
SOLVE EVPORTFOL USING MINLP MAXIMIZING OBJ ;
```

When using DICOPT and CBC it is very important to have the constraints represent to the full extent possible the link between continuous and integer variables.

21.7 Identifying the solver

MIP problems are sometimes hard to solve and sometimes involve trying out different alternatives. Option statements are involved as discussed in the options chapter.

MINLP
MIP
RMIP
RMINLP

21.7.1 MINLP

This option specifies what solver GAMS will use when it needs to solve a MINLP type of model. This option is used by setting

```
Option MINLP=solvername;
```

where the solver must be MINLP capable.

21.7.2 MIP

This option specifies what solver GAMS will use when it needs to solve a MIP type of model. This option is used by setting

```
Option MIP=solvername;
```

where the solver must be MIP capable.

21.7.3 RMIP

This option specifies what solver GAMS will use when it needs to solve a RMIP type of model. This option is used by setting

© 2022 Prof. Bruce McCarl

```
Option RMIP=solvername;
```

Where the solver must be RMIP capable.

21.7.4 RMINLP

This option specifies what solver GAMS will use when it needs to solve a RMINLP type of model. This option is used by setting

```
Option RMINLP=solvername;
```

Where the solver must be RMINLP capable.

21.8 Model termination conditions and actions

The following termination conditions can occur

Value of Modelstat	indicates integer Model Solution Status
8	Integer solution found
9	Solver terminated early with a non-integer solution found(only in MIPs)
10	No feasible integer solution could be found

When a number 9 occurs one may need to consider whether the gap is satisfactory or whether the model has to be run for a longer time. When 10 occurs there truly may be no feasible MIP solution and this can be hard to diagnose. In such cases one should certainly make sure that the RMIP has a feasible solution, then try to fix in an integer solution that should be feasible and find out why not.

21.9 Things to watch out for

There are some problems one may have either due to GAMS settings or problem characteristics. I summarize three of these below.

Default bounds
Ending with a gap – big default for Optcr (10%)
The nonending quest

21.9.1 Default bounds

One needs to be aware that the GAMS default bound used to limit the maximum value of the integer variables to 100 but as of version 23.1 this can be changed to +inf by altering a parameter.

This is controlled using the GAMS parameter IntVarUp=n or option statement option intvarup=n;. as discussed for the command line parameter case here or the option statement case here.

© 2022 Prof. Bruce McCarl

This used to be implemented through a command line parameter pf4.

21.9.2 Ending with a gap – big default for Optcr (10%)

MIPs solves often end with a gap between the solution found and the best possible. This is controlled by optcr/optca or by non convergence. The default value of Optcr is relatively large being 0.10 or a 10% gap as discussed above. Users may want to reduce this to a smaller value. The other cause of a gap is discussed just below.

21.9.3 The nonending quest

Integer programming is a quite desirable formulation technique. But, integer problems can be hard to solve due to search nature of solution process

Three approaches can help

- Reformulate reflecting as much problem knowledge in the formulation as possible improving the depiction of the
  - Way the integer variables are tied together by the constraints. For example, entering constraints that reflect that if one size of machine is chosen in the first stage of an assembly line that it must be matched with a comparable machine in the second stage.
  - Way the integer and continuous are tied together by the constraints. For example, one could achieve benefits by entering constraints indicating that the sum of the continuous variables depicting volume through a warehouse whose construction is depicted by integer variable representing warehouse must be no more than capacity times the integer variable and no less than say 75% of capacity times the integer variable.
  - Restricting the values of the integer variables eliminating "unnecessary" cases of integer variables. For example, in a warehouse location problem if only one warehouse can practically be built require the sum across the variables to be one. Similarly if experience says at least 2 machines are necessary but no more than 5 then enter such constraints.
- Use MIP solver features through options and GAMS. Sometimes spectacular reductions in solution time can be achieved in very little time by changing solver branch and bound procedures.
- Expand available resources or allowable iterations.

Also these solves are generally slower so one must be patient.

22 NLP and MCP Model Types

There are a number of items users should be aware of regarding problem set up, termination conditions and other matters for NLP and MCP problems. Non linear programs or NLPs refer to that class of programs where either the objective function or constraints contained nonlinear terms. Mixed Complementarity Problems or MCPs models can also contains such terms. There is nothing special about the specification of nonlinear terms, as one just uses multiplication(*), exponentiation(**) or division(/) operators within the algebraic specifications of the ..equation contents. One can also use a number of the available functions defined (Exp, Log, Log10, Prod, Sqr, and Sqrt, plus others) as covered in the chapter on calculations.

© 2022 Prof. Bruce McCarl

Terminology
Problem setup
Output
NLP and MCP variants
Solvers

22.1 Terminology

Superbasic
Complementarity

22.1.1 Superbasic

Some of the solvers refer to variables that are superbasic. To solve a linear program it is sufficient to look at basic solutions or "corner points": A set of basic variables (with the number of basic variables equal to the number of constraints) is used to satisfy the constraints, and the remaining so called non-basic variables are all at either their upper or their lower bound (frequently zero).

In nonlinear programming, the optimal solution is usually not at a corner point. Some of the variables will still be at their lower or upper bounds and these variables are again called non-basic variables. The number of remaining variables will usually be larger than the number of constraints. Some solvers will split these remaining variables into basic variables, that are adjusted to satisfy the constraints (the number of basic variables is always equal to the number of constraints), and superbasic variables, that are adjusted to minimize or maximize the objective function. The number of superbasic variables is sometimes called the degrees of freedom. It is a measure of the nonlinearity of the solution around the current point.

22.1.2 Complementarity

Mixed complementarity model types (MCP) require a complementarity between the variables and the equations where either

a variable is nonzero and a constraint strictly binding

or

the constraint is non binding and the variable zero.

They also require the number of variables in the model to exactly match the number of equations in the model. This is discussed below.

22.2 Problem setup

Staring points -- initial values
Upper and lower bounds
Scaling
Degenerate cycling blocking

© 2022 Prof. Bruce McCarl

Advanced bases
MCP complementarity specification

22.2.1 Starting points -- initial values

One strategy one can use to improve solver performance involves use of starting points where initial values are provided for the decision variables within the problem. The specification of starting points involving good initial values for the individual variables is important in a NLP/MCP context for a number of reasons.

- Non-convex models may have multiple solutions and the solvers generally only try to find one local one. An initial point in the right neighborhood is more likely to return a desirable solution.
- Initial values that satisfy or closely satisfy many of the constraints reduce the work involved in finding a first feasible solution.
- Initial values that are close to the optimal ones reduce the work required to find the optimal point and therefore the solution time.
- The progress of the optimization algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and an improved initial point can improve solver performance.

Starting points are specified by assigning values to the level attribute of the variables before solution. Namely, one enters lines like

```
x.l(setdependency)= startingvalue;
```

into the code after the definition of the variables, but before the first solve.

The initial values used within one of the nonlinear solvers employed by GAMS are suggested by GAMS based on the problem formulation and the initial values provided. By default the initial value chosen for all variables is zero or the lower bound. Unfortunately, zero is in many cases is a bad initial value for a nonlinear variable. For example, an initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable. Zero starting values can also cause numerical difficulties when logarithms, exponentiations or divisions are involved. Also nonzero lower bound derived starting points may not be desirable as derivatives evaluated at small lower bounds may be very large and provide the algorithm with misleading information.

Users should endeavor to supply as many sensible initial values for the nonlinear variables as possible by assignment to the level value, var.L, in GAMS as illustrated in the Basis Chapter. It may be desirable to initialize all variables to 1, or to the scale factor if employing the GAMS' scaling option. A better possibility is to select reasonable values for some variables that from the context or from prior solutions using advanced basis concepts like a GDX solution point file.

Constraint relationships can help in providing initial values. For example if the constraints

```
eq(i) .. x(i) = log(a(i)*y(i));
```

were in a model and you have assigned initial values to y.l(i), then you could provide a feasible solution by assigning x.l(i) = log(a(i)*y.l(i)).

© 2022 Prof. Bruce McCarl

One should try to the extent possible to assign initial values to as many as possible of the interrelated variables. Assume you have assigned a value to y.l(i) in the constraint above and have left x.l at zero. Commonly when a solver is trying to find a feasible solution, it will adjust one variable for each constraint. If the solver selects y to be adjusted (a good choice since it is away from its bounds and therefore free to move) then it will change y to the value that makes the constraint feasible and remove your starting point.

22.2.2 Computing Derivatives

Ordinarily in GAMS the derivatives and hessians of NLPs are computed analytically for use by solvers. However GAMS allows extrinsic function libraries that do not have features to return first derivatives and/or Hessian values plus uses numerical derivatives in some cases with the intrinsic function. GAMS approximates these using finite differences.

In particular for a univariate function (F(x)) when a derivative is needed around the point x, GAMS uses a step size parameter d and evaluates the function at f(x-d) and f(x+d) divided by 2d to develop a numerical gradient. In the Hessian computation GAMS can evaluate these based in differences in the first derivative f by computing multiple differences based off the function using f(x-2d), f(x), and f(x+2d). For multi variate functions, the same is done for all the variables involved.

In this procedure users can exercise control over the size of the d parameter (called the step size) using the option **FDDelta.** Also users can control the method of calculation of the Hessian plus cause the first derivative to be computed numerically using **FDOpt.**

Specifically using the option

`FDDelta` One can alter the step size (d) used in the numeric gradient and Hessian calculation as discussed above. The default for `FDDelta` is 1e-5.

`FDOpt` Is used to control step size scaling, hessian computation method and possible numerical first derivative use by specifying two digits: ij.

- Where the first i digit controls whether the step size (FDDelta) is scaled to reflect the size of the value of the point (x) around which the derivative is being formed. When i=0 the default state then scaling is used and d is set to the FDDelta value multiplied by max(1,|x|) (ie. d=max(1,|x|)*FDDelta). When i is set to 1 this turns off scaling and d=FDDelta.

- The j digit is mostly for testing, but allows one to influence gradient computation when the extrinsic function provides gradient but no Hessian values. The numerical derivatives routine in this case uses the gradient calculation from the extrinsic function to approximate the Hessian. If the gradient is expensive to calculate compared to a function evaluation, it could be beneficial to use multiple function values to approximate the Hessian. In this case set the j digit to 1.

© 2022 Prof. Bruce McCarl

- Here are all possible values for FDopt:
  - 0: All derivatives are computed analytically if available, for numerical Hessian use gradient values, FDdelta will be scaled
  - 1: All derivatives are computed analytically if available, for numerical Hessian use function values, scale delta
  - 2: Gradient are computed analytically, force Hessian numerically using gradient values, scale delta (testing only)
  - 3: Gradient are computed analytically, force Hessian numerically using function values, scale delta (testing only)
  - 4: Force gradient and Hessian numerically, scale delta (testing only)
  - 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 (testing only)
  - 13: Same as 3, but no scale of delta (testing only)
  - 14: Same as 4, but no scale of delta (testing only)

These options can be set from the command line, in an option statement, or as a model attribute.

An example without derivatives can be found in library file trilib which implements a Sine function without derivatives leaving their calculation to GAMS.

22.2.3 Upper and lower bounds

Lower and upper bounds in nonlinear models can

- Represent constraints on the reality that is being modeled, e.g. a variable must be positive, or it must be less than or equal to 10.
- Help the algorithm by preventing it from moving far away from any optimal solution and avoiding regions where problems are encountered like unreasonably large function or derivative values as well as singularities in the nonlinear functions. These bounds are called algorithmic bounds.
- Facilitate solution feasibility since most solvers will honor bounds at all times, but inequalities are not necessarily satisfied at intermediate points.
- Improve presolve performance since NLP solvers preprocessors incur a computational penalty when inequalities are present.

Algorithmic bounds merit further discussion. When a model contains the Log or Log10 of a variable it is generally useful to have that variable be no smaller than a given value like 1.e-3 (x.lo=1.0e-3) as the log gets very large as zero is approached and is undefined if the variable becomes negative. Similarly use of Exp of a variable in model equations implies that the variable value should be less than 20-25. Small values for variables used with negative exponents or in denominators are again not desirable. Solver performance can be improved and execution errors avoided when one introduces algorithmic bounds on variables not naturally bounded by the model formulation.

© 2022 Prof. Bruce McCarl

22.2.4 Scaling

Nonlinear programming algorithms use the derivatives of the objective function and the constraints to determine good search directions and to determine how to alter the value of the decision variable levels. They also use function values to determine if constraints are satisfied or not. The scaling of the variables and constraints, i.e. the units of measurement used for the variables and constraints, determines the relative size of the derivatives and function values. They also determine the solution levels and marginals and the search path taken by the algorithm. Proper, consistent scaling of these items is important to the success of the solution algorithm and the quality of the answer returned.

In this context proper consistent scaling means one should scale to achieve:

- Solution level values for the variables fall into a range around 1, e.g. from 0.01 to 100.
- Solution values of the nonzero constraint marginals that exhibit absolute values falling into a range around 1, e.g. from 0.01 to 100.
- Derivatives of nonlinear terms (Jacobian elements) in the model equations that fall in absolute value around 1, e.g. from 0.01 to 100 both at the starting values and at the optimal solution.
- Constants in the model equations that exhibit absolute values around 1, e.g. from 0.01 to 100.

This generally implies that user defined scaling should be employed as discussed in the Scaling GAMS Models chapter. Note some of the solvers have internal scaling procedures, but in general users can do a better job. Scaling factors involve a model level specification of the scaleopt model attribute and an individual variable-by-variable and equation-by-equation specification of the scaling factors.

22.2.5 Degenerate cycling blocking

Most of the commercial linear programming solvers have provisions within them to temporarily place small numbers on the right hand sides of problems if during a solution process they sense that degenerate cycling is occurring. In general, nonlinear programming solvers do not have such an internal feature. Sometimes nonlinear programming solution success and solver performance can be enhanced by making model formulation additions along those lines.

In particular, if users find that the nonlinear programming solution process exhibits a large number of iterations where the solver does not make significant progress in altering the objective function value, then some action may be in order. I have had success with speeding up the nonlinear solutions by altering the zero right hand sides in problems. In particular, sometimes I add a relatively small number to the right hand side of the equation so that instead of

```
f(x) <=  0
```

we have

```
f(x) <= delta*0.001
```

where delta is set to one if we wish the additions and zero otherwise and the 0.001 is adjusted to the size needed for the constraint. Such an addition quite frequently reduces solution time by helping the solver avoid degenerate cycling and if done correctly really does not make a qualitative difference in resultant model solution. One can also set delta to zero after an optimal solution has been achieved and resolve to clean out the effects of the small numbers.

© 2022 Prof. Bruce McCarl

Notes:

- The magnitudes of the numbers added to the right hand side depend on the model context.
  - The numbers should be chosen so they do not introduce significant distortions into the problem solution.
  - One also should not always utilize the same number but perhaps some systematically varying number or a random number.
- Such actions make the constraints easier to satisfy and helps avoid degenerate cycling.
- I not only use this on problems with right hand sides of zero, but also on problems which have a lot of common and equal non zero right hand sides where I will add a small number to the right hand sides.

22.2.6 Advanced bases

In nonlinear models advanced bases are often helpful but there are cases where they do not contribute to shortened solution time. There are no general rules. Only experience with a problem will show how the effectiveness of the provision of an advanced basis. Nonlinear programming is the place where I've had largest gains in terms of solution time reduction with things being reduced from day to hours. Formation and usage of advanced bases are discussed in the Advanced Basis Usage chapter.

Sometimes advanced bases can cause difficulties in nonlinear programming model types. In such cases one may need to manage a model so there aren't radical departures in terms of the number of incorporated variables and equations relative to be saved basis helping avoiding initial singularity. One may also need to use the Bratio option or the model attribute `defPoint` to destroy the basis.

22.2.7 MCP complementarity specification

Mixed complementarity problem model types must obey a property where there are exactly as many variables as there are equations and each variable must be specified as being complementary with one and only one equation. This is done in the Model statement where one lists the equations to be included followed by a period(.) and the name of the associated **complementary variables** as follows (**mcp.gms**)

```
Model qp6 Michael Ferris example of  MCP
/ d_x?x,  d_w?w,  retcon?m_retcon,
budget?m_budget, wdef?m_wdef  /;
```

where the notation has

```
equation name?complementary variable name
```

In addition one can use the /all / notation to automatically include all equations and match up complementary variables. This will work provided that

- All the equations are of the form =e= or =n=.
- The variables are all free (specified as Variables not Positive Variables).

© 2022 Prof. Bruce McCarl

- The problem being square with free variables defined with exactly matching subscripts for each equation.
- This does not fully convey the complementary structure to the solver and is generally inferior to a user defined specification of the complementary pairs.

This is illustrated in **kormcp.gms**.

The MCP problem structure imposes particular requirements on specification of the equations in the problem.

- It is always acceptable to write the equations defining the problem with =N= relations. In this case, the sign of the associated equation is implied by the equation/variable matching and the variable bounds.
- If a variable is complementary with an equation and that variable has lower bounds only (e.g if the variable in GAMS terms is a positive variable) then it is acceptable to write the complementary equation with =G= relations, since this is consistent with the constraint implied by the lower bound on the variable.
- A variable bounded only above can be matched with =L= equations.
- Free variables without bounds can be matched with =E= equations.

These restrictions are elaborated on in the Model Types and Solvers chapter.

22.3 Output

Output files associated with nonlinear terms in models contain several unique characteristics.

Problem displays - limrow/limcol marking
Model setup output
Solver results

22.3.1 Problem displays - limrow/limcol marking

Nonlinear terms are marked in the limrow and limcol output. Namely, nonlinear coefficients are enclosed in parentheses, and the value of the coefficient depends on the level attributes (.l values) of the variables. The listing shows the partial derivative of each variable evaluated at their current level values and there is an implicit unlisted constant involved with the function evaluations. For the example **simpnlp.gms** with the equation

```
Eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2;
```

At the starting point

```
x.l = 2; y.l = 3 ;
```

The equation listing contains

```
Eq1.. (221)*x + (216.1667)*y =2 ; (LHS = 225.5, INFES = 223.5 ***)
```

© 2022 Prof. Bruce McCarl

In this case the 221 against x is the first derivative with respect to x = 2*2*x*power(y,3)+5 (note that the linear term 5*x is treated as part of the nonlinear function), while the 216.1667 against y is the first derivative with respect to y = 2*sqr(x)*3*power(y,2) + 1.5/power(y,2) and the 225.5 is the current evaluation of the function left hand side 2*sqr(x)*power(y,3) + 5*x - 1.5/y and the INFES value is the LHS evaluation minus the RHS evaluation. The proper interpretation of the x and y above are the differences from the starting point thus 221 is the change in the function as x incrementally varies from the starting value of 2.

Note when scaling is present this list will contain the scaled derivatives. The numbers listed are directly useful in evaluation of the scaling criteria mentioned above.

22.3.2 Model setup output

The model setup output as discussed in the Standard Output chapter also has some other features influenced by the solution of nonlinear or MCP model types. Namely for the example **simpnlp.gms** it is

```
MODEL STATISTICS

BLOCKS OF EQUATIONS       1     SINGLE EQUATIONS        1
BLOCKS OF VARIABLES       2     SINGLE VARIABLES        2
NON ZERO ELEMENTS         2     NON LINEAR N-Z          2
DERIVATIVE POOL           6     CONSTANT POOL          11
CODE LENGTH              54
```

This output provides details on the size and nonlinearly of the model.

- The non linear n-z entry refers to the number of nonlinear matrix entries in the model.
- The code length entry reports on the complexity of the nonlinear part of the model. The numerical value reports the amount of code GAMS passes to the nonlinear solver that describes all the nonlinear terms in the model.
- The derivative pool and constant pool provide more information about the nonlinear information passed to the nonlinear solver.

22.3.3 Solver results

Nonlinear problem solutions lead to slightly different reports of solver results. These manifest themselves in the nature of the iteration log, the termination messages that can be encountered, in error reporting and in the MCP case in the nature of certain items in the output.

22.3.3.1 Iteration log

While the iteration log is certainly different when a nonlinear problem is being solved there is no standardized output here. Namely the iteration log is constructed by the solver. Thus the exact output depends on the solver being used and is a user choice. For log file discussion see the solver manuals.

However a general statement can be made. Namely nonlinear programming solution outputs are somewhat more technical as a gradient based approximating process is usually being employed involving the presence of superbasic variables. Generally the solver log output is designed to keep the

© 2022 Prof. Bruce McCarl

user informed on algorithm related factors including such things as the magnitude of the gradients, the step size, and the number of superbasic variables in the solution.

22.3.3.2 Termination messages

Nonlinear solvers may terminate in a number of ways. This section will elaborate where needed on the meaning of the termination messages and in cases point to discussions of corrective actions. This will be done in terms of the reported Model Status or model attribute .Modelstat, and the reported Solver Status or .Solvestat model attribute.

- In general users should recognize that nonlinear programming problem optimal solutions will always be reported as Locally Optimal. This occurs since the solvers and GAMS generally do not try to verify whether the conditions for true global optimality exist (e.g. does the problem involve a globally concave function subject to convex constraint set). That does not mean that the solution found is not a global optimum just that that verification is left to the user.
- Models may be Locally Infeasible. However, this can be misleading. In particular, if the model is non-convex it may have a feasible solution in a different region and a different starting point may reveal a feasible solution.
- Models may terminate with an Feasible Solution being terminated by the solver. Discontinuities, poorly scaled models, or degenerate cycling can cause the problem as can iteration limits, or time limits.
- Models may be unbounded. This occurs if a variable exceeds a maximum value. This can be caused by genuine unboundedness or by poor scaling.

22.3.3.3 Function evaluation errors

The solvers may terminate with numerically based errors. These can involve domain errors when the user defined nonlinear terms are evaluated at solution points that cause under or overflows. In such cases one often has to add algorithmic bounds as discussed above, or engage in scaling as discussed in the Scaling Chapter. Further discussion of such errors appears in the Execution Errors chapter. One can relax the number of these allowed using the Option Domlim. The model attribute domusd gives a count of the number encountered.

22.3.3.4 MCP difference in Equation and Variable Solution Output

Upon solution the MCP generated answer for the equilibrium problem **econequil.gms** appears as below.

```
                            LOWER     LEVEL     UPPER    MARGINAL

---- EQU PDemand             6.000     6.000      +INF      10.000
---- EQU PSupply            -1.000    -1.000      +INF      10.000
---- EQU Equilibri~           .         .         +INF       3.000

                            LOWER     LEVEL     UPPER    MARGINAL

---- VAR P                    .        3.000      +INF         .
---- VAR Qd                   .       10.000      +INF         .
---- VAR Qs                   .       10.000      +INF         .
```

© 2022 Prof. Bruce McCarl

This does not have an objective function but otherwise is of basically the same form as typical GAMS output. However there are some interpretation differences.

In the equation output four columns appear for each equation. GAMS transforms each constraint so that the terms involving variables are collected on the left hand side, and accumulates constants on the right hand side. The lower and upper bounds give the resultant constants. For equality equations, these are equal, whereas for inequalities one is infinite. The level value is an evaluation of the left hand side of the constraint at the current point. The model may be infeasible and if so the unsatisfied equations are marked with INFES. The Marginal column gives the value of each variable complementary to the constraint. It is also possible to obtain output indicating equations have been redefined (REDEF). A technical explanation of this phenomenon appears in the Path manual (**PATH**) but it is just a warning and mandates no user action.

In the variable listing, four output items are again provided. The lower, and upper items give the lower and upper bounds on the variables. The level gives the solution value. The marginal column contains the value of the slack on the equation that was paired with this variable. The definition of this slack is the minimum of equ.level - equ.lower and equ.level - equ.upper, where equ is name of the paired equation. Note that the variable levels and the equation marginals correspond.

A summary report follows that indicates how many errors were found. When the model has infeasibilities, these can be found by searching for the string "INFES"

More on this can be found in the Path manual.

22.4 NLP and MCP variants

Nonlinear and MCP model types also encompass the forms MCP, NLP, CNS, MINLP, and DNLP models as well as MPSGE and MPEC models. These are discussed in the Model Types and Solvers chapter.

22.5 Solvers

These model types require specialized solvers. The applicable solvers and the nonlinear model types they apply to (note the solvers can also solve linear cases).

The GAMSIDE provides a list of all solvers available and an indication which model types they apply to.

Using the following statement in a GAMS model will also produce a list of solvers and the corresponding model types:

```
option subsystems;
```

23 Model Attributes

Model attributes contain information relative to model solution procedures and results passed back and forth between GAMS and solvers.

There are two fundamental types of model attributes:

© 2022 Prof. Bruce McCarl

- Attributes mainly used before a solve statement to provide information to GAMS or the solver link.
- Attributes mainly used after a solve statement to retrieve information about the last solve.

The adressing of model attributes and the use of single attributes will be discussed below.

Attribute addressing
Model Attributes mainly used before solve
Model Attributes mainly used after solve

23.1 Attribute addressing

Model attributes are addressed with modelname.attribute i.e.

```
X=modelname.attribute;
Modelname.attribute=3;
```

where modelname is the name used in a model statement and attribute is one of the items listed below. More specifically, given the modelname is transport then statements like

```
x=transport.Modelstat;
transport.holdfixed=1;
transport.bratio=1;
```

can be specified. Such items

- can be used in displays
- can be used in calculations as data (on the right hand side of assignments)
- may have calculation results assigned to them (on the left hand side of equations)

23.2 Model Attributes mainly used before solve

Attributes mainly used before a solve statement provide information to GAMS or the solver link. Beside the standard attribute addressing some of these input attributes can also be set via an option statement or the command line

```
option reslim = 10
gamsmodel.gms reslim = 10
```

The third and fourth column of the following table indicate if this ways of setting are possible. Note that a model specific option takes precedence over the global setting and a setting via an option statement takes precedence over the setting via the command line.

Attribute	Description	option	command line

© 2022 Prof. Bruce McCarl

bRatio	Basis acceptance test. A bratio of 0 accepts any basis, and a bratio of 1 always rejects the basis. See Advanced Basis Usage.	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	Cutoffvalue. 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.		
defPoint	Determines the point that is passed to the solver as a basis. By default, the levels and marginals from the current basis is passed to the solver. In some circumstances (mostly during debugging) it can be useful to pass a standard default input point, i.e. with all levels set to 0 or lower bound. When defpoint=1 this results in passing zoro or lower bounf levels and zero marginals. When defpoint=2 this results in curent variable and equation levels set by the user and defaults (0) for the marginals.		
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
fddelta	Step size when computing numerical derivatives	x	x
fdopt	Controls method of computing numerical Hessian	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
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	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	x	x

© 2022 Prof. Bruce McCarl

	0.1.		
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 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 modelname p.gdx is to be saved from the last solve in the GAMS model • 2: a point gdx file called modelname 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. See Scaling GAMS Models.	x	x
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 modelname.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 (default) • 2/Silent: suppress all solution information	x	x
solveLink	This attribute controls GAMS function when linking to solve. Allowable values are: • 0: GAMS operates as always (default) • 1: the solver is called from a shell and GAMS remains open	x	x

© 2022 Prof. Bruce McCarl

	• 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.		
solveOpt	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 model name.solveOpt and the command line expect a numeric value. Allowed are: • 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 • 1/merge: the solution information for all equations and variables is merged into the existing solution information; (default) • 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 There is an example called 'solveopt' in the model library.	x	
sysOut	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 model name.sysOut and the command line expect a numeric value. Allowed are • 0/Off: suppress additional solver generated output (default) • 1/On: include additional solver generated output	x	x
threads	This attribute controls the number of threads or CPU cores to be used by a solver. Allowable values are: • -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)	x	x
tolInfeas	Infeasibility tolerance for an empty row of the form a.. 0*x =e= 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'.		
tolInfRep	This attribute sets the tolerance for infeasible solutions. The default value is 1.0e-6.		
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.		

© 2022 Prof. Bruce McCarl

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 model name.trylinear=1. 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. default value is 1.0		x
workSpace	This attribute tells the solver how much workspace in Megabytes to allocate for problem solution.		x

23.3 Model Attributes mainly used after solve

The model attributes mainly used after solve are set by GAMS or the solver link and contain information about the last solve. These attributes are automatically reset during execution of a solve.
These attributes can be accessed by

X=modelname.attribute;

as described in Attribute Addressing.

Attribute	Description
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 handlecollect, handlestatus or handledelete. See Grid Computing language features
iterUsd	Number of iterations used.
line	Line number of last solve of the corresponding model.
linkUsed	Integer number that indicates the value of solveLink used for the last solve.

© 2022 Prof. Bruce McCarl

maxInfes	Max of infeasibilities as reported by a solver
meanInfes	Mean of infeasibilities as reported by a solver
modelStat	Integer number that indicates the model status. Possible values are: 1 Optimal solution achieved 2 Local optimal solution achieved 3 Unbounded model found 4 Infeasible model found 5 Locally infeasible model found (in NLPs) 6 Solver terminated early and model was infeasible 7 Solver terminated early and model was feasible but not yet optimal 8 Integer solution model found 9 Solver terminated early with a non integer solution found (only in MIPs) 10 No feasible integer solution could be found 11 Licensing problem 12 Error achieved – No cause known 13 Error achieved – No solution attained 14 No solution returned from CNS models 15 Feasible in a CNS models 16 Locally feasible in a CNS models 17 Singular in a CNS models 18 Unbounded – no solution 19 Infeasible – no solution For more information see Modelstat: Tmodstat
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.
numVar	Number of variables.

© 2022 Prof. Bruce McCarl

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. Possible values are: 1 Normal termination 2 ran out of iterations (fix with iterLim) 3 exceeded time limit (fix with resLim)) 4 quit with a problem (see LST file) found 5 quit with excessive nonlinear term evaluation errors (see LST file) 6 terminated for unknown reason (see LST file) 7 The solver cannot find the appropriate license key needed to use a specific subsolver. 8 User interrupt 9-13 terminated with some type of failure (see LST file) For more information see Solvestat: Tsolstat
sumInfes	Sum of infeasibilities.

© 2022 Prof. Bruce McCarl

23.3.1 Marginals

This is a model attribute that indicates whether (1) or not (0) a dual solution is returned from the solver after the solve statement.

When the solver does not provide a dual solution (.m), then GAMS does not print the marginal column in the solution listing and set the marginal field in variables and equations to NA.

23.3.2 Modelstat: Tmodstat

Upon solution a model attribute indicative of model status (Modelstat) is set by a solver and passed back to GAMS. This attribute can have the following values

Value of Modelstat	Indicated Model Solution Status	ModelStat Constant
1	Optimal solution achieved	ModelStat.Optimal
2	Local optimal solution achieved	ModelStat.Locally Optimal
3	Unbounded model found	ModelStat.Unbounded
4	Infeasible model found	ModelStat.Infeasible
5	Locally infeasible model found (in NLPs)	ModelStat.Locally Infeasible
6	Solver terminated early and model was infeasible	ModelStat.Intermediate Infeasible
7	Solver terminated early and model was feasible but not yet optimal	ModelStat.Feasible Solution
8	Integer solution model found	ModelStat.Intermediate Non-Integer
9	Solver terminated early with a non integer solution found (only in MIPs)	ModelStat.Integer Infeasible
10	No feasible integer solution could be found	ModelStat.Integer Infeasible
11	Licensing problem	ModelStat.Licensing Problem
12	Error achieved – No cause known	ModelStat.Error Unknown
13	Error achieved – No solution attained	ModelStat.Error No Solution
14	No solution returned	ModelStat.No Solution Returned
15	Feasible in a CNS models	ModelStat.Solved Unique
16	Locally feasible in a CNS models	ModelStat.Solved
17	Singular in a CNS models	ModelStat.Solved Singular
18	Unbounded – no solution	ModelStat.Unbounded - No Solution
19	Infeasible – no solution	ModelStat.Infeasible - No Solution

Notes:

- Also the older labeling for condition 7 - ModelStat.Intermediate NonOptimal has a numerical value of 7

© 2022 Prof. Bruce McCarl

- These values are fixed and cannot be manipulated.

In addition, the model attribute .Tmodstat contains internally assigned text describing the model optimality status and can be used in put files to print out a text indication of model optimality status.

Examples:

(varmodatt.gms)

```
scalar xx;
xX=transport.Modelstat;
Display xx,transport.Modelstat;
If(transport.Modelstat gt %ModelStat.Locally Optimal%,
      Display '**** Model did not terminate with normal solution',
                  transport.Modelstat;)
set scenarios /goodone,toomuchdemand/;
set casewithbadanswer(scenarios,*);
loop (scenarios,
      solve transport using lp minimizing cost;
      demand(sink)=demand(sink)*1.1;
      casewithbadanswer(scenarios,"model")
           $(transport.Modelstat gt 2)=yes;
);
display casewithbadanswer;
```

Notes:

- In the example above note we are basically using this attribute so I can keep a record or generate output relative to model solution status.
- The user can assign values to Modelstat, but ordinarily this is not desirable.
- In a set of looped solves this provides the best way to keep track of whether the solves worked properly and if the numerical values are stored to record an indication of the type of problem.
- .Tmodstat contains internally assigned text describing the model optimality status and can be used in put files to print out a text indication of that status.

23.3.3 Solvestat: Tsolstat

Upon solution a model attribute indicative of solver termination model status (solvestat) is set by a solver and passed back to GAMS. This attribute can have the following 13 values.

Value of Solvestat	Indicated Solver termination condition	SolveStat Constant
1	Normal termination	Solvestat.Normal Completion
2	Solver ran out of iterations (fix with iterlim)	Solvestat.Iteration Interrupt
3	Solver exceeded time limit (fix with reslim)	Solvestat.Resource Interrupt
4	Solver quit with a problem (see LST file) found	Solvestat.Terminated By

© 2022 Prof. Bruce McCarl

Value of Solvestat	Indicated Solver termination condition	SolveStat Constant
		Solver
5	Solver quit with excessive nonlinear term evaluation errors (see LST file and fix with bounds or domlim)	Solvestat.Evaluation Interrupt
6	Solver terminated for unknown reason (see LST file)	Solvestat.Capability Problems
7	Solver terminated with preprocessor error (see LST file)	Solvestat.Licensing Problems
8	User interrupt	Solvestat.User Interrupt
9	Solver terminated with some type of failure (see LST file)	Solvestat.Setup Failure
10	Solver terminated with some type of failure (see LST file)	Solvestat.Solver Failure
11	Solver terminated with some type of failure (see LST file)	Solvestat.Internal Solver Failure
12	Solver terminated with some type of failure (see LST file)	Solvestat.Solve Processing Skipped
13	Solver terminated with some type of failure (see LST file)	Solvestat.System Failure

These values are fixed and cannot be manipulated.

Examples:

(varmodatt.gms)

```
scalar xx;
xX=transport.solvestat;
Display xx,transport. solvestat;
If(transport.solvestat gt %Solvestat.Normal Completion%,
      Display '**** Solver did not terminate with normal solution',
                  transport. solvestat ;)
set scenarios /goodone,toomuchdemand/;
set casewithbadanswer(scenarios,*);
loop (scenarios,
         solve transport using lp minimizing cost;
         demand(sink)=demand(sink)*1.1;
         casewithbadanswer(scenarios,"solve")
$(transport.solvestat gt 1)=yes;
);
display casewithbadanswer;
```

Notes:

- In the example above this attribute is used to keep a record of termination status and generate output relative to solver termination status.
- The user can assign values to solvestat, but ordinarily this would not be the case.

© 2022 Prof. Bruce McCarl

- In a set of looped solves this provides another way to keep track of whether the solves worked properly and if the numerical values are stored to record an indication of the type of problem.
- Tsolstat contains internally assigned text describing the solver termination status and can be used in put files to print out a text indication of termination status.

24 Application Help: Model Library, Web Sites, Documentation

GAMS has been used in a lot of settings with many different approaches and procedures used. While the material above was designed to cover GAMS in a relatively comprehensive manner there are certainly other sources of information that users will find useful particularly when pursuing specific type of modeling exercises. Here I provide a guide to additional resources one can pursue in a quest for more information on GAMS.

Model library
Other general documentation sources

24.1 Libraries

GAMS has been an important vehicle for modeling for more than a decade. As such there are numerous applications that have implemented. Examination of previous applications allows the user to examine approaches and techniques that have been tested by others gaining or adapting ideas for the application at hand. To facilitate such an exercise GAMS Corporation has assembled a library of models, collectively called GAMSLIB.

The models in the library have been selected because they represent interesting and sometimes classic problems or illustrate GAMS techniques. Problems in the library depict such diverse applications as

- Production and shipment by firms,
- Investment planning in time and space,
- Cropping pattern selection in agriculture,
- Operation of oil refineries and petrochemical plants,
- Macroeconomics stabilization,
- Applied general equilibrium analysis,
- International trade in commodities,
- 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.

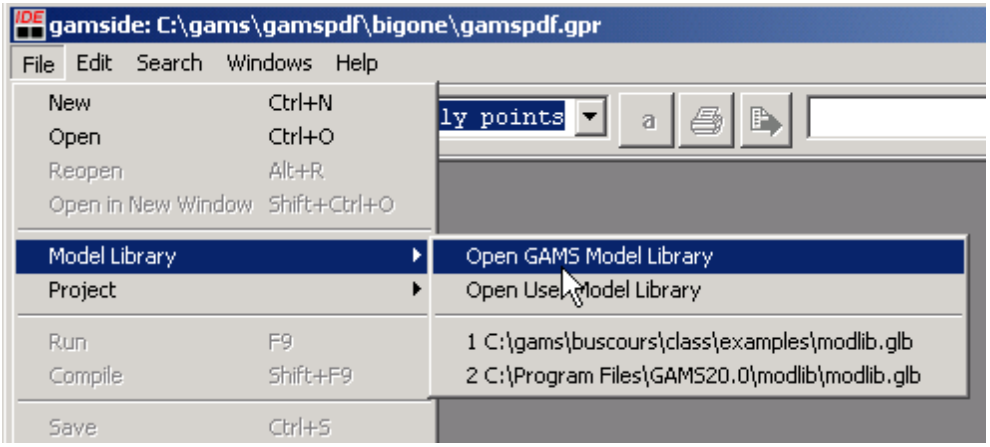
© 2022 Prof. Bruce McCarl

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.

Using the GAMS model library
Using another model library

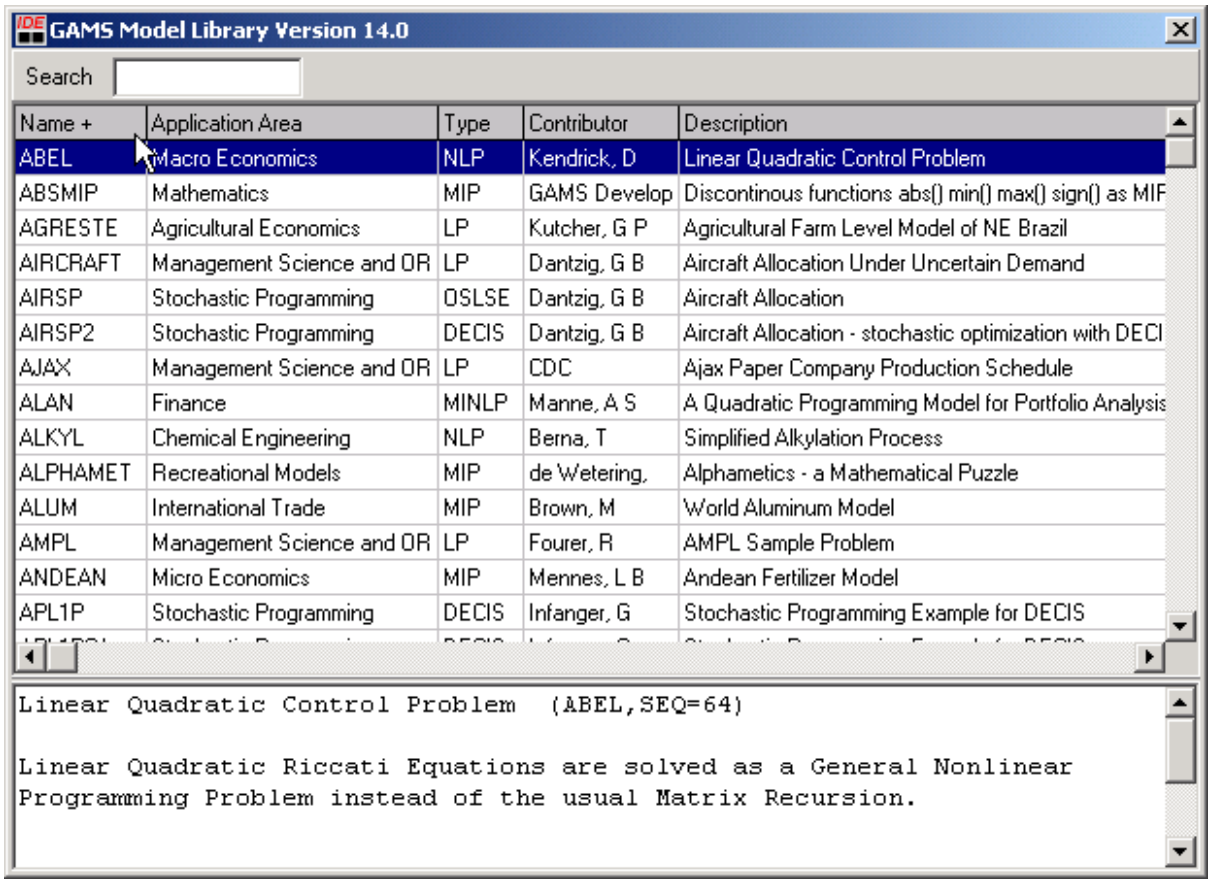
24.1.1 GAMS model library

The library is most conveniently accessed through the IDE. In particular a Windows Explorer like library manager is present in the IDE. You invoke this by invoking the File menu, choosing the Model Library line and the Open GAMS Model Library choice



This invokes the Library manager that, once opened, brings up the following screen

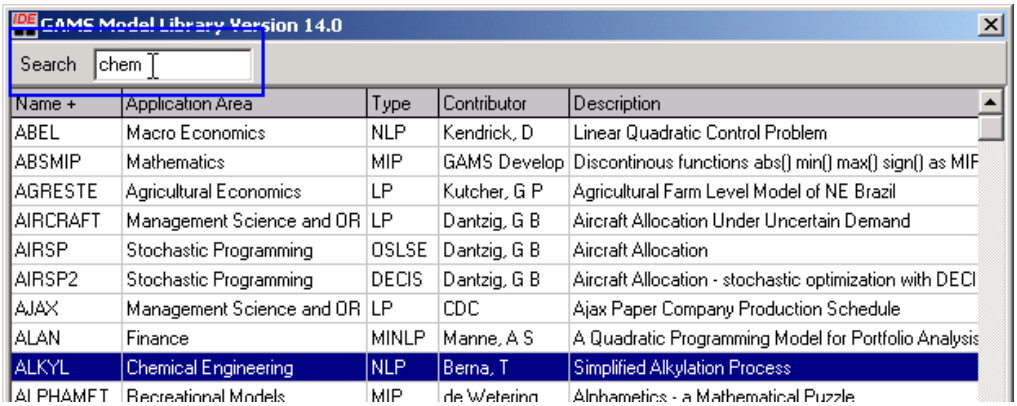
© 2022 Prof. Bruce McCarl



The grid filled with model names is a scrollable list of files with column entries describing file attributes. At the bottom is a more lengthy description of the file highlighted. In turn double clicking in a row causes the file to be loaded in the IDE for editing and into your project directory. Simultaneously all files it includes and other files the library creator nominated are placed in the project directory.

There are several features of this manager that merit description

- The manager contains a search box as in the blue box below. When one types an entry into that box the manager automatically scrolls forward to a file that contains the typed text somewhere in it. For example typing in the string **"chem"** causes the screen below to appear



© 2022 Prof. Bruce McCarl

where the ALYKL command is the first one that contains the string **"chem"**. If one wants to find the next occurrence of the string one can press the down arrow key while the up arrow reveals the previous one.

- The gray bar at the top of the grid allows one to alter the tabular sort order. By clicking the mouse on the gray bar one changes that order. For example touching the word Type causes the files to be sorted by model type as illustrated below.

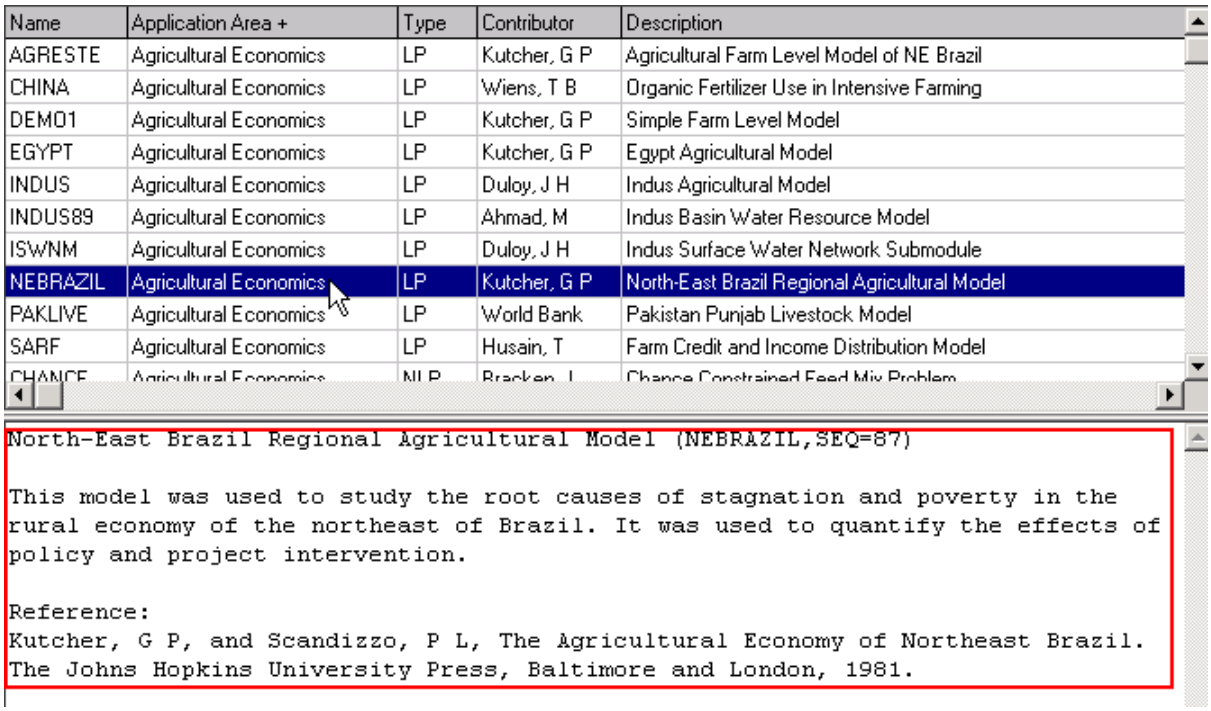
Name	Application Area	Type +	Contributor	Description
CAMCNS	Applied General Equilibrium	CNS	Condon, T	Cameroon General Equilibrium Model Using CNS
GANCNS	Applied General Equilibrium	CNS	Mitra, P	Macro-Economic Framework for India - CNS
GANCNSX	Applied General Equilibrium	CNS	Mitra, P	Macro-Economic Framework for India - Tracking CNS
KORCNS	Applied General Equilibrium	CNS	Lewis, J	General Equilibrium Model for Korea - CNS
AIRSP2	Stochastic Programming	DECIS	Dantzig, G B	Aircraft Allocation - stochastic optimization with DECIS
APL1P	Stochastic Programming	DECIS	Infanger, G	Stochastic Programming Example for DECIS
APL1PCA	Stochastic Programming	DECIS	Infanger, G	Stochastic Programming Example for DECIS
FARM	Stochastic Programming	DECIS	Birge, J R	The Farmer's Problem formulated for DECIS
PRODSP2	Stochastic Programming	DECIS	King, A J	Stochastic Programming Example - reformulated for DECIS
LINEAR	Econometrics	DNLP	Bracken, J	Linear Regression with Various Criteria
WATER	Engineering	DNLP	Brooke, A	Design of a Water Distribution Network
REPAY	Finance	GAMS	GAMS Develop	Repayment Factors for Loans
CALENDAR	GAMS Language Features	GAMS	GAMS Develop	Calendar Function Examples
GAMSHTM	GAMS Language Features	GAMS	GAMS Develop	HTML generation of model library
GAMSUTIL	GAMS Language Features	GAMS	GAMS Develop	Generates some useful files from the Model Library

The column sorted on is marked with a + if in forward order and a minus if in reverse order.

The sort order also uses a secondary sort key where it remembers the previous columns you have sorted on. Consequently if you first click on Name then Type you the files will be arranged by alphabetical order of their names under a problem type. But if you sorted first on Application Area then most recently on Type they would be organized by Application Area under each problem type.

- A left click with the mouse on a file name causes its description to be placed in the text box at the bottom. A double click loads the file into the IDE.

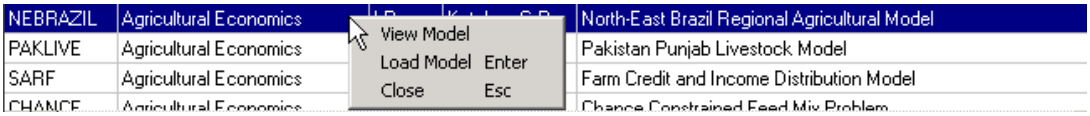
© 2022 Prof. Bruce McCarl



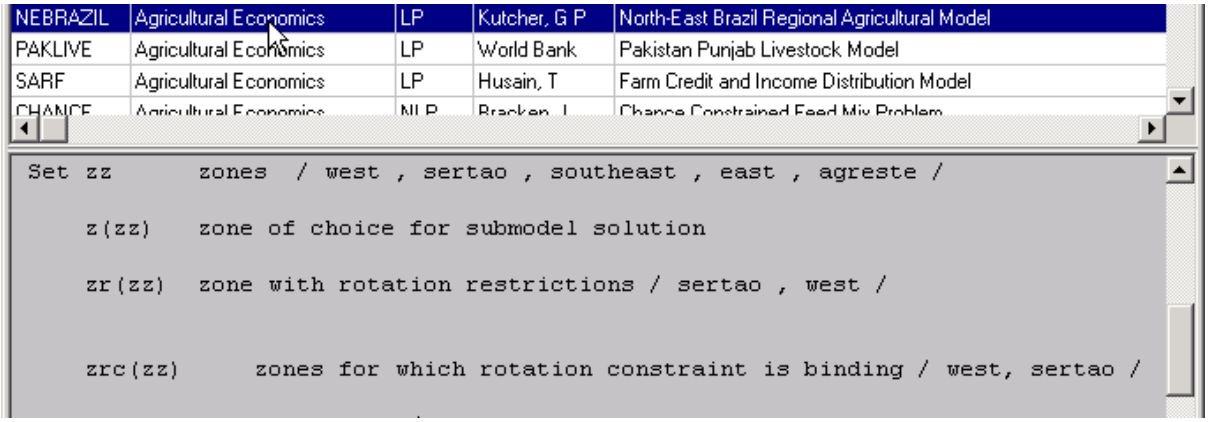
Name	Application Area +	Type	Contributor	Description
AGRESTE	Agricultural Economics	LP	Kutcher, G P	Agricultural Farm Level Model of NE Brazil
CHINA	Agricultural Economics	LP	Wiens, T B	Organic Fertilizer Use in Intensive Farming
DEMO1	Agricultural Economics	LP	Kutcher, G P	Simple Farm Level Model
EGYPT	Agricultural Economics	LP	Kutcher, G P	Egypt Agricultural Model
INDUS	Agricultural Economics	LP	Duloy, J H	Indus Agricultural Model
INDUS89	Agricultural Economics	LP	Ahmad, M	Indus Basin Water Resource Model
ISWNM	Agricultural Economics	LP	Duloy, J H	Indus Surface Water Network Submodule
NEBRAZIL	Agricultural Economics	LP	Kutcher, G P	North-East Brazil Regional Agricultural Model
PAKLIVE	Agricultural Economics	LP	World Bank	Pakistan Punjab Livestock Model
SARF	Agricultural Economics	LP	Husain, T	Farm Credit and Income Distribution Model
CHANCE	Agricultural Economics	NLP	Bracken, J	Chance Constrained Feed Mix Problem

```
North-East Brazil Regional Agricultural Model (NEBRAZIL,SEQ=87)

This model was used to study the root causes of stagnation and poverty in the
rural economy of the northeast of Brazil. It was used to quantify the effects of
policy and project intervention.

Reference:
Kutcher, G P, and Scandizzo, P L, The Agricultural Economy of Northeast Brazil.
The Johns Hopkins University Press, Baltimore and London, 1981.
```

- A right click with the mouse brings up a menu box.



If one chooses the view model choice the GAMS code is placed in the box where the description appeared with a gray background.



NEBRAZIL	Agricultural Economics	LP	Kutcher, G P	North-East Brazil Regional Agricultural Model
PAKLIVE	Agricultural Economics	LP	World Bank	Pakistan Punjab Livestock Model
SARF	Agricultural Economics	LP	Husain, T	Farm Credit and Income Distribution Model
CHANCE	Agricultural Economics	NLP	Bracken, J	Chance Constrained Feed Mix Problem

```
Set zz      zones  / west , sertao , southeast , east , agreste /

    z(zz)   zone of choice for submodel solution

    zr(zz)  zone with rotation restrictions / sertao , west /


    zrc(zz)     zones for which rotation constraint is binding / west, sertao /
```

Another right click will bring up the box again and the view model choice can be unclicked.

- You may change column widths, allocation of top and bottom screen parts and window size just as in other programs through use of the mouse. The IDE will remember **some but not all** of your choices.
- Users may define their own library by using a GLB file as discussed here.

© 2022 Prof. Bruce McCarl

24.1.2 GAMS Test Library

This is a library of GAMS models that is used by GAMS Corporation for quality control and testing. Most users will not have a use for this library. It is accessed through the Model Libraries choice on the IDE menu.

24.1.3 GAMS Data Utilities Library

A library of files that illustrates passing data to and from GAMS. Illustrates ASK, GDXXRW, and a number of other data passing routines. It is accessed through the Model Libraries choice on the IDE menu.

24.1.4 GAMS EMP Library

A library of files that illustrates use of the GAMS EMP facility. Illustrates ASK, GDXXRW, and a number of other data passing routines. It is accessed through the Model Libraries choice on the IDE menu.

24.1.5 GAMS Financial Library

This library (FINLIB) is an alphabetical listing of the models available in the on-line model library based on the book by A. Consiglio, S.S. Nielsen and S.A. Zenios, Practical Financial Optimization: A library of GAMS models, John Wiley, UK, 2009. The models range from simple cashflow matching models to several variants of Markowitz mean-variance optimization to advanced models for international asset allocation and currency hedging, corporate bond portfolio management, asset and liability modeling for insurers as well as for individual investors, and the management of indexed funds. Some documentation on the library is at https://www.gams.com/latest/finlib_ml/libhtml/index.html.

It is accessed through the Model Libraries choice on the IDE menu.

24.1.6 Using another model library

Users may address other Library's provided they have been built and supplied by others or the user goes through the exercise of creating one and defining a GLB file. For details see the following

How the library works: http://www.gams.com/mccarl/uselib.pdf.

How to create libraries: http://www.gams.com/mccarl/createlib.pdf.

A program that creates a library: makeindx at http://www.gams.com/mccarl/makeindx.zip .

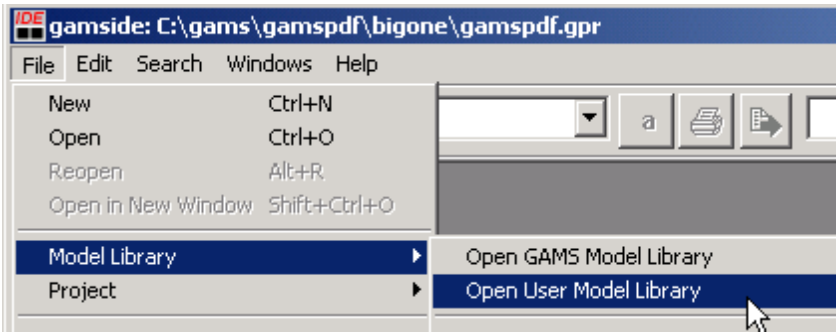
A full example library from McCarl and Spreen book: http://www.gams.com/mccarl/textprob.zip. (for the book itself see http://agecon2.tamu.edu/people/faculty/mccarl-bruce)

All the files for library creation are in the directory with the examples for this manual. The library is built from the $Ontext/$Offtext entries that appear at the bottom of each example referenced herein along

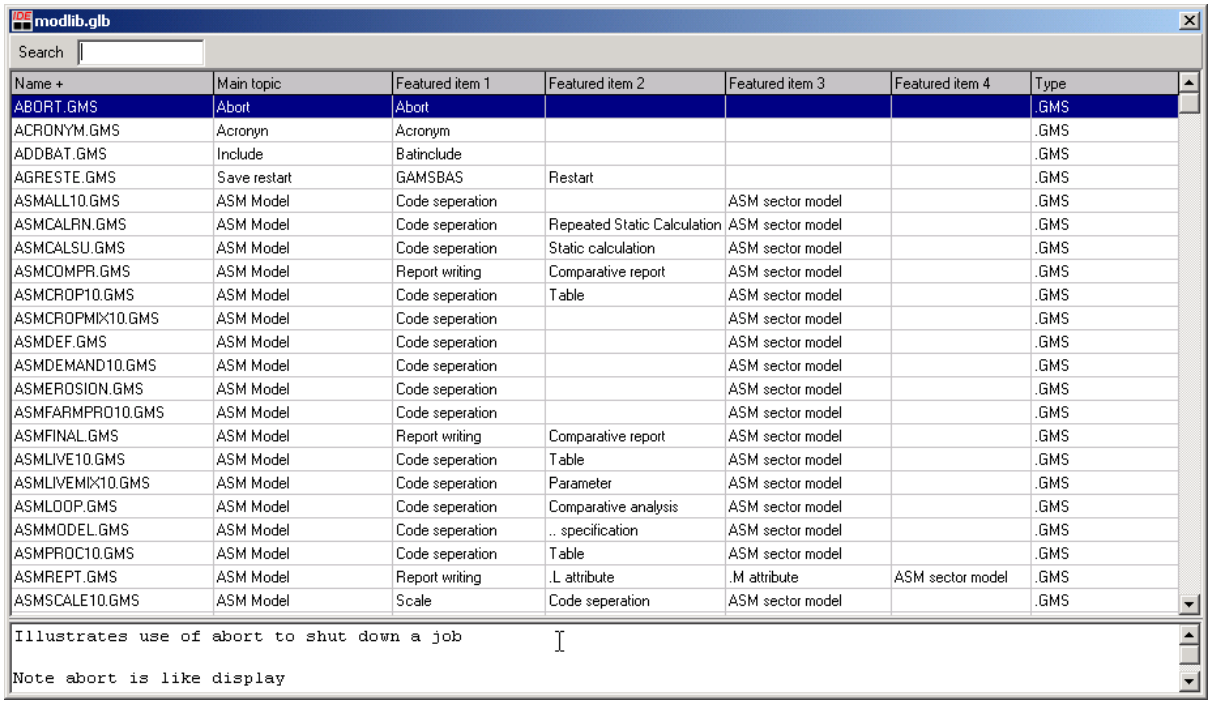
© 2022 Prof. Bruce McCarl

with the file gamslib1.ini and makeindx.exe.

Such a library has been built for most of the examples used in this manual. You may access that library through a 2 step process. First you need to inform the IDE about the presence of this library by using the File option and the model library choice then the Open user model library choice.



In turn you can browse for the directory where a file called modlib.glb resides which will be in the directory where all of the example problems for this manual appear and double click on that file. Subsequently the library manager opens and you get a screen as follows



Name +	Main topic	Featured item 1	Featured item 2	Featured item 3	Featured item 4	Type
ABORT.GMS	Abort	Abort				.GMS
ACRONYM.GMS	Acronyn	Acronym				.GMS
ADDBAT.GMS	Include	Batinclude				.GMS
AGRESTE.GMS	Save restart	GAMSBAS	Restart			.GMS
ASMALL10.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMCALRN.GMS	ASM Model	Code seperation	Repeated Static Calculation	ASM sector model		.GMS
ASMCALSU.GMS	ASM Model	Code seperation	Static calculation	ASM sector model		.GMS
ASMCOMPR.GMS	ASM Model	Report writing	Comparative report	ASM sector model		.GMS
ASMCROP10.GMS	ASM Model	Code seperation	Table	ASM sector model		.GMS
ASMCROPMIX10.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMDEF.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMDEMAND10.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMEROSION.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMFARMPRO10.GMS	ASM Model	Code seperation		ASM sector model		.GMS
ASMFINAL.GMS	ASM Model	Report writing	Comparative report	ASM sector model		.GMS
ASMLIVE10.GMS	ASM Model	Code seperation	Table	ASM sector model		.GMS
ASMLIVEMIX10.GMS	ASM Model	Code seperation	Parameter	ASM sector model		.GMS
ASMLOOP.GMS	ASM Model	Code seperation	Comparative analysis	ASM sector model		.GMS
ASMMODEL.GMS	ASM Model	Code seperation	.. specification	ASM sector model		.GMS
ASMPROC10.GMS	ASM Model	Code seperation	Table	ASM sector model		.GMS
ASMREPT.GMS	ASM Model	Report writing	.L attribute	.M attribute	ASM sector model	.GMS
ASMSCALE10.GMS	ASM Model	Scale	Code seperation	ASM sector model		.GMS

This Library has all the examples in the book and sortable columns giving topics and GAMS commands used. The 4 columns Features Item 1-4 give specific commands in the examples but contain overlapping content. Thus one can find references to the same item in all 4 columns. One thus when questing for help on a command like sameas needs to look across all 4 columns or use the search box.

One may also load this library into the IDE by augmenting the idecfg.ini file.. In particular In

© 2022 Prof. Bruce McCarl

particular one can alter the file idecfg.ini so it has contents like

```
[library1]
text=GAMS Model Library
file=modlib\modlib.glb

[library2]
text=GAMS Test Library
file=gtestlib\testlib.glb

[library3]
text=McCarl GAMS Classes Library
file=mccarlgams\example\modlib.glb
```

where the blue content is there by default and the red content is ones additions. In turn today the glb file for the library and all the library files must be located in the relative path given by the file=location command which would be a subdirectory of the GAMS system directory (C:\program files\GAMS22.7 on a US machine). This will change in the next release

24.2 Other general documentation sources

In addition to the material herein there are a number of other sources for documentation and modeling tips. These are listed and briefly discussed below.

Installation
Latest GAMS version
Solver manuals
GAMS FAQ
GAMS World
Gams-List
Newsletter
Supplemental GAMS Corporation materials
User generated materials
Courses and workshops

24.2.1 Installation

When installing GAMS one needs machine specific instructions on the steps to follow. Unix and PC based instructions are

- On paper in the source documents sent with the system.
- On the distribution CD.
- In the GAMS system directory under the subdirectory DOCS.
- Accessible through the web on on the GAMS web site (*http://www.gams.com/*) in the *Documentation (including FAQ)* area.
- Accessible through the IDE Help function as explained in the Running Jobs with GAMS and the GAMS IDE chapter.

© 2022 Prof. Bruce McCarl

24.2.2 Latest GAMS version

GAMS maintains a set of release notes and a release history on its web page. The release history is on the main page at *https://www.gamsworld.org/*. The release notes are clickable links associated with each release date although to date they are cumulative so by looking at the last one can see the developments since the version that the user has.

There is also an automatic email based service with which you can discover what updates are available to GAMS above and beyond those in your current system. It will also tell you whether your license file is recent enough to allow these updates. You invoke the system as described on *https://www.gams.com/latest/docs/UG_License.html*.

- Arrangements can be made to download the latest version through *http://www.gams.com/download/* although one cannot use it unless their license file is current as discussed in the Model Types and Solvers chapter.

24.2.3 Solver manuals

Users wishing details on solver procedures and options will need to consult the solver reference guides. These guides are

- On paper in the source documents sent with the system.
- On the distribution CD.
- In the GAMS system directory under the subdirectory DOCS.
- Accessible through the web on the GAMS web site (*http://www.gams.com/*) in the *Documentation (including FAQ)* area.
- Accessible through the IDE Help function as explained in the Running Jobs with GAMS and the GAMS IDE chapter.

24.2.4 GAMS FAQ

GAMS provides a web list of Frequently Asked Questions and answers at *http://www.gams.com/docs/FAQ/index.htm*. Contents are currently present on general GAMS issues, modeling issues, computer system (platform) issues and solver issues. A sample of the types of questions addressed appears below

- GENERAL
  - Solver option file not found (wrong file extension)
  - How can I access environment variables inside GAMS
  - Command Line Parameters
  - The Solver Option File
  - Special MIP features
  - GAMS/Virus Scanner incompatibility
  - How can I enumerate all possible subsets of a set
  - How do I reduce the size of my listing (.LST) file?

© 2022 Prof. Bruce McCarl

- MODELING
  - Missing trig functions (arccos, arcsin, tan)
  - Sensitivity analysis (parameter ranges) for LP
  - How do I model piecewise linear functions?
  - Smooth approximations for MAX(X,0) and MIN(X,0)
  - MIN function, don't use it
- PLATFORM
  - GAMS crash after upgrade to Windows 2000
  - How do I run GAMS in the background on a UNIX machine?
  - Finding GAMS errors using the vi editor
  - Executing GAMS using the run command in Windows
  - How to use GAMS from a Unix shell
  - How to execute GAMS from different directories
- SOLVER
  - Why do various MIP solvers give differing answers to my problem?
  - How do I interrupt a solver?
  - GAMS/Virus Scanner incompatibility
  - Adding a constraint to a model and do a warm start
  - How to use option files

24.2.5 GAMS Wiki

GAMS has a support Wiki that contains some material collected from support activities. The wiki page is mainly designed as internal databases for their support staff but has been made accessible in the hope that users will find it valuable.

24.2.6 GAMS World

GAMS World is a web site at *http://www.gamsworld.org/* designed to "bridge the gap between academia and industry by providing highly focused forums and dissemination services in specialized areas of mathematical programming". The web site currently covers

- Global nonlinear optimization World addressing computational methods to find global optimal solutions to nonconvex nonlinear optimization problems on *http://www.gamsworld.org/global/index.htm*. The coverage involves solvers, a library of test problems, a list server mailing list and a page of links.
- Mixed Integer Nonlinear Programming (MINLP) World addressing computational methods to find solutions to MINLP type problems on *http://www.gamsworld.org/minlp/index.htm*. The

© 2022 Prof. Bruce McCarl

coverage involves solvers, a library of test problems, a list server mailing list and a page of links.

- Mathematical Programs with Equilibrium Constraints (MPEC) World addressing computational methods to find solutions to MPEC type problems on *http://www.gamsworld.org/mpec/index.htm*. The coverage involves a library of test problems, a list server mailing list and a page of links.
- Mathematical Programming System for General Equilibrium (MPSGE) World addressing economic equilibrium modeling with the Mathematical Programming System for General Equilibrium (MPSGE) on *http://www.gamsworld.org/mpsge/index.htm*. The coverage involves a library of test problems, a list server mailing list and a page of links.
- Performance World addressing performance testing of mathematical programming problems on *http://www.gamsworld.org/performance/index.htm*. The coverage includes a set of test problems, automation tools for collecting performance measurements, and tools for analyzing and visualizing test results.
- Translation World addressing translation of GAMS models to other formants based on the "solver" GAMS/CONVERT. The coverage involve an email translation server.

24.2.7 GAMSWorld Google Group

A discussion group associated with https://www.gamsworld.org/, a web site aiming to bridge the gap between academia and industry by providing highly focused forums and dissemination services in specialized areas of mathematical programming. Users may subscribe at http://groups.google.de/group/gamsworld .

24.2.8 Gams-List

GAMS users worldwide use a mailing list named GAMS-L to exchange information about GAMS. One can subscribe through the GAMS World Page at *https://forum.gamsworld.org/viewtopic.php?f=16&t=11619*.

The pdf file for the old archive can be retrievable at https://forum.gamsworld.org/download/file.php?id=2010.

The new archive is at https://www.listserv.dfn.de/sympa/arc/gams-l.

24.2.9 Newsletter

I issue a newsletter that is designed to provide additional information on the use and features that emerge as GAMS develops. Short newsletters appear sporadically, when events justify, containing information on new developments and/or under documented features as well as opportunities to learn more about GAMS usage. Users may subscribe to Bruce McCarl's GAMS newsletter by filling out a form found at *https://www.gams.com/newsletter/signup/*.

© 2022 Prof. Bruce McCarl

24.2.10 Supplemental GAMS Corporation materials

GAMS provides a number of its own and a set of contributed documents on the web site *http://www.gams.com*.

24.2.11 User generated materials

A number of users have web sites where GAMS materials appear. These include

- Tom Rutherford
  - Tools site at *http://www.mpsge.org/inclib/tools.htm*
  - Web interfaces he developed or helped with on GAMS interfaces at http://www.mpsge.org/gamssm/index.html
  - MPSGE materials at *http://www.gams.com/solvers/mpsge/*, *http://www.gamsworld.org/mpsge/*, and *http://www.mpsge.org/mainpage/mpsge.htm*
- Michael Ferris
  - MATLAB related site *http://www.cs.wisc.edu/math-prog/matlab.html*
  - Complementary problem net *http://www.neos-guide.org/content/complementarity-problems*
- Rob Dellink
  - GAMS materials at *http://www.enr.wur.nl/uk/gams/*
- Bruce McCarl
  - Agricultural modeling book with Tom Spreen at *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/regbook.htm*
  - GAMS materials at *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/gamsmcc.htm*
  - Documents in support of newsletter *http://www.gams.com/docs/contributed/index.htm*
- Materials from a number of others as listed at
  - *http://www.gams.com/docs/contributed/index.htm*
  - *http://www.gams.com/contrib/contrib.htm*
  - *https://www.gams.com/archives/presentations/*

24.2.12 Courses and workshops

A number of short courses catering to general users and users with specific interests are offered. These are announced on *http://www.gams.com/courses.htm*

25 Compressed and encrypted files

Models may be developed for distribution but the model developer may wish to limit access to parts of the model to insure privacy, security, data/model integrity and ownership. To address this one may use

- **Compressed and Encrypted Input Files** or

© 2022 Prof. Bruce McCarl

- can compress and encrypt GAMS input files as discussed below.

Input file compression and decompression are available to all users. Encryption and secure work files require special licensing. Three Dollar Control Options control this:

$Compress	<source> <target>	Causes compression of an input file
$Decompress	<source> <target>	Causes decompression of an input file
$Encrypt	<source> <target>	Causes encryption of an input file

Notes

- 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 same license as was used in encryption.
- Once a file has been encrypted it cannot be decrypted.
- Decompression and decrypting is done when reading the GAMS system files.
- GAMS recognizes whether a file is compressed or encrypted and will process the files accordingly.
- All compressed and encrypted files are platform independent.

Following the GAMS example in the document File "Compression and Encryption" that is now Appendix I in the old GAMS user guide, the model library file TRNSPORT is used to illustrate compression. First suppose we compress the file using the following GAMS commands as in **compress.gms**

```
$call 'gamslib trnsport'
$compress trnsport.gms t2.gms
```

where the first command retrieves the transport file and the second command compresses it.

Then later we can solve it as follows as in **compress.gms**

```
$include t2.gms.
```

treating the compressed input file like any other GAMS input file. Note here the list file will look just as it did with the uncompressed file. To hide the listing of parts one can insert $Offlisting and $Onlisting around the blocks of GAMS code that are not to be echoed.

One can decompress the file and recover the original file as in **decompress.gms**

```
$Decompress t1.gms. t4.gms
```

One can similarly encrypt a file by using

```
$encrypt trnsport.gms t1.gms t2.gms
```

© 2022 Prof. Bruce McCarl

With the command line parameter plicense=target

The compressed version of t1.gms can only be used with the target license file.

The cefiles.gms model from the GAMS Model Library contains a more elaborate example on compression while the encrypt.gms model from the GAMS Model Library contains a more elaborate example on encryption.

26 Grid and Distributed Computing

The GAMS language has been extended to take advantage of systems with multiple CPUs and threads plus High Performance Computing Grids allowing model solutions tasks to be carried out in parallel. One can also distribute jobs across multiple CPUs in a much less automated way. In both cases this allows model solutions tasks to be carried out in parallel.

To see more on this see the sections on

Grid Computing

Distributed Processing

Language features that implement grid/multi threading computing are discussed next.

Users may supply a script to submit jobs using the gridscript option.

Further documentation on the use of this feature appears in the Grid computing section and in the document here.

26.1 Distributed Processing

GAMS model instances can take a long time to solve and one may need to solve the model for a number of scenarios with the result taking weeks or months.

GAMS has the GRID computing facility but on PCs use over a network of machines requires substantial IT involvement. Such an implementation is often impractical. Here we present a semi automated procedure to accomplish this.

© 2022 Prof. Bruce McCarl

Basically, the code (merge_control_example.gms) writes GAMS script files to run a distributed set of jobs on multiple processors/machines involving creating GDX files to pass scenario data and return scenario results plus writing a GAMS file that runs the scenarios. The procedure also compresses the files for the remote machines into a zip file for transfer. A run of merge_control_example.gms also writes a file merge_mergethem.gms that merges all into a unified file and a script file that runs it.

An example was written that used the AGRESTE.GMS model library file and is available here.

The core code - merge_control_example.gms creates all files used in the whole process. It is particular to runs with the AGRESTE case.

The example extends the AGRESTE.GMS model library file and is available here.

The core code - merge_control_example.gms creates all files used in the whole process. It is AGRESTE context specific with AGRESTE related data handling, reporting and scenario development but also many generic features. Within that code the user tells: a) how to allocate scenarios to processors, and b) what data to save and load. Then it writes script files for each processor plus a processor specific GDX file containing the scenario definition data. It also writes a file that runs the scenarios. In turn it zips the all the relevant files for transfer to the remote machine.

On the distributed processors/machines the user needs to move in the zip files and extract then then initiate the scrip file whereupon the scenarios are run and as they finish the results are automatically unloaded into GDX files. Then the user needs to transfer these GDX files back to a location on which the data will be merged by running merge_mergethem.gms.

Operationally in the main merge_control_example.gms file we

1. Define the scenarios and in this agreste example link them to some alternative manipulations of the agreste price, land and risk aversion data (see section 1 in the code).
2. Define the processors to be used and assign scenarios to be run by each processor (see section 2)
3. Give a name to the procedure that will run the scenarios on the assigned processors (in this case agreste_loop - see section 3)
4. Identify which report parameter that reflect the solution for each scenario are to be merged (see section 4).
5. Set up parameters for the data items and sets that define the scenarios (see section 5)
6. Create processor specific GDX files that contain scenario data to be run and processor specific script files plus create a zip file to pass to the processor (Section 6).
   There are a number of subparts to this step. In particular, for each processor we
   a. Set up scenario data and put it in an appropriately named gdx file (section 6a)
   b. Create a script file for that processor (section 6b)

© 2022 Prof. Bruce McCarl

7. Create GAMS code in the file merge_mergethem.gms that will merge the results (section 7).
   Here there are several steps involved
   a. We define a control variable that contains the names of the report items to merge - section 7a.
   b. We declare the parameters and sets that are involved with holding the scenario dependent reports to be merged (not needed if restarting) - section 7b.
   c. We include a statement to load the gdx file data from each processor using $loadm so the data will be merged - section 7c.
   d. We form a set that tells what scenarios have been found - section 7e.
   e. We display merged results and write a gdx file of them - section 7f
8. Get agreste.gms from the library and run it to create a restart file - section 8.
9. Create a GMS file that runs the scenarios called agreste_loop - section 9. In doing this we
   a. Set up scenario particulars - section 9a
   b. Define the names of the cross scenario reports that are computed and subsequently passed on for use in the merge operations - section 9b.
   c. Load in data that tells what scenarios to use and accompanying data - section 9c.
   d. Loop through the scenarios assigned to a processor setting it up, solving it and computing reports on the results - section 9d.
   e. Unload data into the gdx file for subsequent use in the merge operation - section 9e

B. In doing this merge control_example.gms writes several types of files

1. Files for use on each of the remote processors
   a. A script file for that processor with a name like script_for_secondprocessor.gms that when run executes all the GAMS tasks needed to be done on that processor. One will be generated for each processor identified in the runstodo table.
   b. A GDX file for each processor with the scenario data named send_to_processorname.gdx or in the example send_to_secondprocessor.gdx
   c. A gms file that is used on all remote processors to runs the scenarios called agreste_loop.gms.
   d. A restart file that is created after agreste is run (a1.g00).
   e. A zip file that contains the 4 files just above for each processor called zip_to_send_to_processorname.zip or in the example zip_to_send_to_Secondprocessor.zip.
2. A file for the merge operation named merge_mergethem.gms that merges the data from the distributed runs assuming the gdx files of results have been moved to the project directory.

C. On the remote processor one needs to

1. Copy in and unzip the file zip_to_send_to_processorname.zip. In this example one file is named zip_to_send_to_processorname.zip. This unpacks the script file (script_for_second_processor.gms), a1.g00, the agreste_loop.gms file and the

© 2022 Prof. Bruce McCarl

send_to_processorname.gdx (e,g, send_to_secondprocessor.gdx)

2. Run the script file which causes a load of the gdx file of data, a run of agreste_loop.gms executing all assigned scenarios, creation of scenario dependent reports and an unloading of the results in a gdx file for use in the merge exercise. (results_processorname.gdx or in the example results_secondprocessor.gdx)
3. Move the results gdx file to the location where the merge is to be run

D. Then back on the main machine

1. One makes sure the results_procssorname.gdx files have been moved.
2. Run the merge_mergethem.gms to obtain the merged results. These will appear in merge_mergethem.lst or in results_agreste.gdx.
3. Note this procedure can be run at any time even if all the gdx files are not present.

A few notes

- The GAMS command **execute_unload** is used to place the results into GDX files since they only exist at execution time.
- The code for loading from the GDX files uses the GAMS command **$loadm** to merge the results as that command augments the earlier information merging in the data.
- merge_mergethem when executed does the merger and it can be followed by something else to create user reports. For this you may want to restart it from an agreste , agreste_loop or merge_control_example restart file.
- Much more could be done with this. For example, one could insert commands that copy the zip file to the remote processors on a network, then unzip and run it. Additionally one could write code to copy back the GDX files of results. One can also write a procedure that waits for the GDX files to be present before running the merge_mergethem.gms file.
-

The example file is available here.

26.2 Introduction to Grid Computing

GAMS allows one to formally solve models across a grid of computers or using multi threading.

The formal grid features work on all GAMS platforms and have been tailored to many different environments. Here we overview the Grid computing features. For those wishing more information and examples see the GAMS web site coverage in the document http://www.gams.com/docs/gridcomputing.htm.

The multi threading features are invoked using Solvelink=6 and are supported by the option ThreadsAsync with documentation here.

Both the grid and multi threading implementations use the general grid computing features as will be

© 2022 Prof. Bruce McCarl

discussed below.

The grid and multi threading facility separates the solution into several steps which are controlled separately. First note that when GAMS encounters a solve statement during execution it proceeds in three basic steps:

- **Generation**. The symbolic equations for a model are used to specify a numerical counterpart of the model using the current data. GAMS generates files containing all information needed by a solution method in a form independent of the solver and computing platform.

- **Solution**. The numerical model and associated files are handed over to a solver and GAMS waits until the solver terminates.

- **Solution Collection Read and Update Loop**. The solutions of the previously submitted models are collected from the solver and put into the internal GAMS data base as they become available. It may be necessary to wait for some solutions to complete.

Language features that implement grid computing are discussed next.

Users may supply a script to submit jobs using the gridscript option.

Further documentation on the use of this feature appears in the Grid computing section and in the document here.

26.2.1 Grid Computing language features

Grid computing involves several functions, a model attribute, a specialized gdx load procedure and the GAMS option GridDir plus the concept of a handle.

Grid Computing : Invoking using Solvelink Model Attribute

The model attribute **.solvelink** tells GAMS when Grid computing is to be used altering the behavior of the solve statement. A value of '**3**' tells GAMS to generate and submit the model for solution and continue without waiting for the completion of the solution step as illustrated above.

```
 set pp names of all models to be solved /m1*m100/;
parameter h(pp) model handles;
 minvar.solvelink=3;  or 6
 Loop(p(pp),
        ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
        Solve minvar min var using miqcp;
        h(pp) = minvar.handle );
```

© 2022 Prof. Bruce McCarl

A set of solvelink constants may also be used.

Handles

A handle in the grid environment identifies the particular model and data instances available. The model attribute **.handle** contains the unique identification data for each submitted solution request and is typically stored in a parameter. Such handles are associated with a set that covers all model instances and are stored in a parameter defined over that set. Their specific numerical values are assigned by GAMS and in turn can be used to recover solutions and manage models being solved on the Grid.

```
parameter h(pp) stored model identifying handles;
minvar.solvelink=3;  or 6
Loop(p(pp),
        ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
        Solve minvar min var using miqcp;
        h(pp) = minvar.handle );
```

Retrieving and Storing Grid solutions

One must interrogate the solution process to see if the models sent to the grid have been solved and then load in the solutions. This is typically done with the function handlecollect(handle). Users are responsible for storing grid solutions in GAMS parameters if they want to be able to separately address the different solutions. This is typically done by defining new parameters for storing necessary variable/equation levels and marginals. Such parameters typically include the model identifying set that is associated with the handles.

One can employ the function ReadyCollect(handleParameter [, maxWait]) to find out when models have finished solving and are ready to be collected. This can be used for both SolveLink = % solveLink.Async Threads% and SolveLink = %solveLink.Async Grid%. The arguments and return codes used with it are

handleParameter: which is a parameter holding handles of model instances to eventually collect

maxWait: which is an optional parameter giving the maximum time to wait which defaults to +INF if omitted.

On return the ReadyCollect function provides a code which is

- 0: if at least one of the requested jobs is complete and ready to be collected
- 1 if there is no active job to wait for
- 2 if no handle was provided
- 3: if an invalid handle was provided

© 2022 Prof. Bruce McCarl

- 4: if the user specified time-out was reached when using a SolveLink = % solveLink.Async Threads% handle
- 5: if the user specified time-out was reached when using a SolveLink = % solveLink.Async Grid% handle
- 8: if an unknown error occurred. this should not happen

If the models take a long time to solve then it is likely that when we go to collect the solution that it may not be ready and will not be retrieved. This means we need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. One can do this using a repeat-until construct as shown below:

```
Repeat
        loop(pp$handlecollect(h(pp)),
                xlevelres(i,pp) = x.l(i);
                ylevelres(i,pp) = xi.l(i);
                rmarginalres(i,pp) = r.m(i) );
                display$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;
```

This also deletes completed models and sets variables equal to na when the model was not solved during the active job time (specified by timeelapsed>100).

Solutions can also be retrieved using GDX files and Handlestatus where one also must identify the specific handle name to be retrieved and use a slecial gdx load syntax

```
Repeat
        loop(pp$(handlestatus(h(pp)=2),
                minvar.handle = h(pp);
                execute_loadhandle minvar;
                xlevelres(i,pp) = x.l(i);
                ylevelres(i,pp) = xi.l(i);
                rmarginalres(i,pp) = r.m(i) );
                display$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;
```

If the models take a long time to solve then it is likely that when we go to collect the solution that it may not be ready and will not be retrieved. This means we need to call this loop several times until

© 2022 Prof. Bruce McCarl

all solutions have been retrieved or we get tired of it and quit. One can do this using a repeat-until construct as shown below:

```
Repeat
        loop(pp$handlecollect(h(pp)),
                xlevelres(i,pp) = x.l(i);
                ylevelres(i,pp) = xi.l(i);
                rmarginalres(i,pp) = r.m(i) );
                display$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;
```

This also deletes completed models and sets variables equal to na when the model was not solved during the active job time (specified by timeelapsed>100).

Solutions can also be retrieved using GDX files and Handlestatus where one also must identify the specific handle name to be retrieved and use a slecial gdx load syntax

```
Repeat
        loop(pp$(handlestatus(h(pp)=2),
                minvar.handle = h(pp);
                execute_loadhandle minvar;
                xlevelres(i,pp) = x.l(i);
                ylevelres(i,pp) = xi.l(i);
                rmarginalres(i,pp) = r.m(i) );
                display$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;
```

Retrieving and Storing Grid solutions

One must interrogate the solution process to see if the models sent to the grid have been solved and then load in the solutions. This is typically done with the function handlecollect(handle). Users are responsible for storing grid solutions in GAMS parameters if they want to be able to separately address the different solutions. This is typically done by defining new parameters for storing necessary variable/equation levels and marginals. Such parameters typically include the model identifying set that is associated with the handles.

© 2022 Prof. Bruce McCarl

Handlestatus constants

A related set of HandleStatus constants is defined

Handlestatus.Unknown is a constant that equals 0.
Handlestatus.Running is a constant that equals 1.
Handlestatus.Ready is a constant that equals 2.
Handlestatus.Failure is a constant that equals 3.

These may be used as follows

```
loop(pp$(handlestatus(h(pp)=%Handlestatus.Ready%),
```

These constants are defined at compile time and cannot be manipulated or reassigned.

Execute_loadhandle for Grid Solution Retrieval from GDX

Using the statement

Execute_loadhandle ***mymodel*;**

causes GAMS to load the model with the current attribute mymodel.handle from a GDX file (provided it has been placed there by use of the Handlestatus function) into the GAMS data base. The command operates otherwise like the execute_loadpoint procedure.

Grid related Functions

Four functions are used to manage the problems in the grid

HandleCollect(handle) tests to see 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 associates with handle had not yet finished solution or could not be loaded
- 1 if the solution has been loaded

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.

HandleStatus(handle) tests to see 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

© 2022 Prof. Bruce McCarl

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 no 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.

HandleSubmit(handle) resubmits a previously created instance of the model identified by the handle for solution.

- 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.

Grid Model Attributes

Three model attributes are associated with the Grid features

***mymodel*.solvelink** specifies tells GAMS how to behave with respect to the solver

- 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
- 4 start the solution and wait (same submission process as 3)

This is used by specifying

***mymodel*.solvelink=3;**

or

***mymodel*.solvelink=0;**

A set of solvelink constants may also be used.

***mymodel*.handle** gives or specifies the current instance handle

This is used to either

- retrieve the handle for a specific model passed to a solver.

  currenthandle= ***mymodel*.handle;**

- identify the handle of the specific model to be retrieved from the GDX file using **execute_loadhandle**.

© 2022 Prof. Bruce McCarl

```
i=HandleStatus(currenthandle);
if(i=2,
        mymodel.handle=currenthandle;
    execute_loadhandle mymodel;
        loop(pp$(savedhandle(pp)=currenthandle),
            xlevelres(i,pp) = x.l(i);
            ylevelres(i,pp) = xi.l(i);
            rmarginalres(i,pp) = r.m(i) );
);
```

***mymodel*.number** gives a current instance number for the model generated by a solve statement

Any time a solve is attempted for *mymodel*, the instance number is incremented by one and the handle is updated accordingly. The instance number can be reset by the user which then resyncs the handle.

Storage location

The grid models for solution and their corresponding solutions are kept in a Grid directory. Each GAMS job has only one Grid Directory. By default, the grid directory is assumed to be the scratch directory. Useds can change this 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 GAMS job will be terminated.

Grid output control

The **AsyncSolLst** option which is also a command line parameter controls whether or not solver output from problems solved using the grid or multi-threading procedures is passed to the LST file. When it is set to one the output goes to the LST file and when set to zero it does not. The default value is zero.

26.2.2 Grid_example

GAMS allows you to solve multiple problems using the multiple cores available on your computer. This is covered under the Grid computing part of the Expanded User Guide as accessed through the IDE help menu. An example is provided the file trnsgrid.gms which is a small rewrite of the model library file with the same name (with a small modification in the definition of repy)

```
parameter repx(s,i,j) solution report
          repy(s,solveinfo) summary report;
repy(s,'solvestat') = na;
```

© 2022 Prof. Bruce McCarl

```
repy(s,'modelstat') = na;
*we use the handle parameter to indicate that the solution has been
collected
repeat
   loop(s$handlecollect(h(s)),
      repx(s,i,j) = x.l(i,j);
      repy(s,'solvestat') = transport.solvestat;
      repy(s,'modelstat') = transport.modelstat;
      repy(s,'resusd'   ) = transport.resusd;
      repy(s,'objval')    = transport.objval;
      display$handledelete(h(s)) 'trouble deleting handles' ;
      h(s) = 0 ) ;    // indicate that we have loaded the solution
      display$sleep(card(h)*0.2) 'was sleeping for some time';
until card(h) = 0 or timeelapsed > 10;  // wait until all models are loaded
display repx, repy;
```

In this code

- Loop solves each problem where the set s identifies those problems
- Repx saves the solution values of the variable x (note in general all the problem variables would need to be saved).
- Repy saves overall solution information on the problem (solution termination status, model optimality etc status, time used and objective function value)
- This code would use each of the multiprocessor cores as they become available.

In general this feature lets you use multiple machines but can take a lot of setup to network the machines particularly in a Windows environment.

27 Interfacing from other languages with API

GAMS offers a low level application programming interface (API) to various programming languages. Recently GAMS has introduced an object oriented (OO) API that allows to control GAMS from within different programming languages like C#, Java and Python.

The first implementation was done for Microsoft's .NET framework and its numerous programming languages (including C#). GAMS Release 24.0 introduced the Java and Python versions of this API.

The OO GAMS API offers a seamless integration of GAMS into .NET, Java and Python based IT systems. The API extends the capabilities of GAMS by the addition of the features provided by these frameworks.

The in-memory representation of a GAMS model can increase performance when solving a model multiple times with slightly changed data by performing the model creation only once.

Convenient data structures allow random access to data as well as data iteration. Collectively this OO GAMS API allows the smooth integration of GAMS into applications by providing appropriate classes for the interaction with GAMS.

A detailed description of the OO APIs can be found at the following places:

.NET API Reference Manual

© 2022 Prof. Bruce McCarl

Java API Reference Manual

Python API Reference Manual

These documents also reside in the /docs/API subdirectory of the GAMS system directory or can be accessed through the IDE help under Docs then API.

In addition to these APIs, there exist more technical, lower level APIs that permit access to GAMS information from the programming languages C, C++, C#, Delphi, Fortran, Java, Python, and Visual Basic. These are overviewed with examples in GAMSx API, GDX API, and Option API. Additional information for GDX interfacing is available in CHM and PDF forms under the GAMSIDE help under API and EXPERT-LEVEL.

28 Licensing

Use of GAMS for all but very small problems requires purchase of rights to use the software in the from of a License file.

There are two types of items included in the license file
- Rights to use the overall GAMS system which is required in all cases
- Rights to use an individual solver in association with the GAMS system which is a user choice

On purchase of a file one gets rights to use the software and all versions developed in the next year. Those versions may be used forever. Users then pay maintainence on their licenses to extend them for versions developed beyond that initial year. the utility CHK4UPD can be used to see if a newer GAMS version is available and what the newest version is that can be used with a given license. This is done by running it without arguments or by giving the name of a local license file.

License files are typically limited to a single computer platform type (e.g. Windows or AIX or LINUX). But as of version 23.6 GAMS introduced a generic license which supports use on all platforms on which GAMS runs (ie it can be employed on more than one type of computer operating system – such as Windows and LINUX). Pricing and details can be gotten from GAMS sales.

GAMS and its solver partners offer a free academic license to members of the solver partners academic initiative programs. Details are available through https://support.gams.com/solver:academic_programs_by_solver_partners

Licenses can be obtained for single and multi user situations.

© 2022 Prof. Bruce McCarl

Index

- -

'
Symbol surrounding set element name or explanatory text that must appear in pairs. 54
Symbol surrounding set element name or text, rules for using 232

- - -

- 11
LST FIle Navigation Window 69, 71, 241
Remove equations when specifying a model 69, 71, 88, 241
Symbol for subtraction. 241
Symbol that is operator to form set difference. 71
Symbol that signifies a lag operation in a set. 69
--
Command line parameter that allows definition of a control variable. 363
Symbol that signifies a circular lag operation in a set. 70
User defined command line parameter. 659

- " -

"
Symbol surrounding set element name or explanatory text that must appear in pairs. 54
Symbol surrounding set element name or text, rules for using 232
"%name%"
Syntax to retrieve named control variable, command line parameter or system attribute and treat result as text. 645

- # -

#
Put file command to skip to a row in put file. 595
Use in defining sets 62, 595

- $ -

$
$$ variant to allow $ to begin in columns other than one 60, 217, 289, 311, 500, 501, 510
Marker for number of compiler error message. 217
Symbol that sets of a compiler time option. 500
Symbol to set off conditionals, use in calculation. 289
Symbol to set off conditionals, use in set references. 60
Symbol to set off conditionals. 311
When dollar commands are executed. 501
$$
Allow $ to begin in Columns other than 1 510
$Abort
.noerror to not increase error count 510, 663
Dollar command that can be used in conditional compilation to issues an error message in LST file. 663
Dollar command that causes compilation to stop and issues an error message in LST file. 510
$Batinclude
Dollar command that includes an external file with arguments. 510
Include an external file with arguments. 494
Repeating use of same code with substituted arguments. 650
$Call 267
Changing parameters when calling 510, 549, 660, 693, 699
Dollar command that executes a program during compilation. 510
Dollar command to cause a compile time run of a command or program. 660
Executing an external procedure at compile time. 693
Timing of execution. 699
$Call =
Rearranging placement of rows and columns when writing from Gdxxrw into spreadsheets. 693
$Call.Async
Dollar command that allows ansynchronous job handling 511, 660
$Call.AsyncNC
Dollar command starting asyncronous jobs 512
$Clear

© 2022 Prof. Bruce McCarl

$Clear
Dollar command that resets named items to default values. Use of the option command clear is usually preferable. 512
$Comment
Change character used to start a comment in column 1 which is now an * 235
Dollar command that changes character used to start a comment in column 1 which is now an *. 513
$Decompress
Decompresses GAMS files 513
$Dollar
Dollar command that resets character that starts dollar option commands. 514
$Double
Dollar command that starts double spacing of echo print lines in LST file. 514
Starts double spaced echo print. 137
$Drop
Dollar command to drop a contol variable 544, 629
$Dropenv
Dollar comamnd to drop a control variable that was defined with $Setenv 547
$Dropglobal
Dollar command to drop a global control variable 546, 628
$Droplocal
Dollar command to drop a local control variable 547, 629
$Echo
Dollar command that echoes text to a named file. 514
Dollar command to echo text to a file. 662
$Echon
Dollar command that echoes multiple lines of text to a named file. 514
$Eject
Dollar command that starts a new page in LST file. 515
Starts a new page in LST file. 162
$Else
Dollar command that is paired with $Ifthen 254, 522
$Elseif
Dollar command that is paired with $Ifthen 522, 640
$Elseife
Numerical value evaluating variant of $Eleseif 522, 640
$Elseifi
Case insensitive variant of $Eleseif 522, 640
$Endif
Dollar command that is paired with $Ifthen 522, 640
$Eolcom
Change symbol for end of line comments 236
Dollar command that changes symbol for end of line comments. 515
$Error
Dollar command that causes reporting of compiler error to LST file but allows continued compilation. 516, 663
$Escape
Dollar command that causes redefinition if % symbol to set off control variables. 516
$Eval
Evaluates numerical scoped contol variable expression 516, 632
$EvalGlobal
Evaluates numerical global contol variable expression 517, 630
$EvalLocal
Evaluates numerical local control variable expression 517, 631
$Exit
Dollar command that exits compilation 517
Dollar command to exit a compilation. 663
$Expose
Dollar command that removes privacy restrictions. 517
$FuncLibIn
Dollar command to bring in function library 508
Including user defined functions 268
$Gdxin
Compile time GDX file naming, opening and closing 673
Dollar command that opens/closes a GDX file for input 518
$Gdxout
Compile time GDX file naming, creation and closing 672
Dollar command that opens/closes a GDX file for output 519
Problems with compile time write to GDX 672
$Goto
Compile time transfer of compiler position. 656

© 2022 Prof. Bruce McCarl

$Goto
Dollar command that transfers control to a line with an internal label. 519
$Hidden
Dollar command that inserts one line comment that does not appear in LST file. 520
Insert one line comment that does not appear in LST file 239
$Hide
Dollar command that hides the objects in a privacy setting but allows them to be used in model calculations. 520
$If
Dollar command that causes a statement executed in compiler if conditional is true. 520
Dollar command that is compile time if test - case sensitive. 637
$If Not
Dollar command that executes a GAMS command in compiler if conditional is false. 520
$Ife
Dollar command that executes a GAMS command if condition is true (expressions in condition ar evaluated) 520
Form of control variable conditional involving numbers 637
$Ife not
Dollar command that executes a GAMS command if condition is false (expressions in condition ar evaluated) 520
$Ifi
Dollar command that executes a GAMS command in compiler if conditional is false and is case insensitive. 520
Dollar command that is compile time if test - case insensitive. 637
$IFTHEN
Dollar command that controls multiple statments 522, 640
If - Endif alternative 656
$ifthene
Numerical expression evaluating variant of $Ifthen 254, 522
$iftheni
Case insensitive variant of $ifthen 522, 640
$Include
Dollar command that includes an external file without arguments. 523
Include an external file without arguments. 491
$Inlinecom
Change character strings setting off in line comment 237
Dollar command that changes character strings setting off in line comment. 523
$Kill
Dollar command that removes all data for an item. This should not be used. 523
$Label
Dollar command compile time label for GOTO jump. 656
Labels a line allowing branching with $goto. 524
$Libinclude
Dollar command that includes a file with arguments from inclib subdirectory. 524
Include a file with arguments from inclib subdirectory. 497
$Lines
Dollar command that starts new page if less than n lines are left on a page. 524
Starts new page if less than n lines are left on a page. 162
$Load
Compile time read from GDX file element identification 673
Dollar command that loads data from a GDX file 524
Listing GDX file contents 680
Using to infer sets 525, 673, 680
Variant with domain checking 524, 673, 680
$Loaddc
GDX Load with domain checking 526
$Loaddcm
Dollar command that loads data from a GDX file merging the contents 526
$Loaddcr
Dollar command that loads data from a GDX file (mixture of $Loaddc and $Loadr) 527
$Loadm
Dollar command that loads data from a GDX file and merges it 527
$Loadr
Dollar command that loads data from a GDX file and replaces sets or parameters with the data from that file 527
$Log 165
Dollar command that sends specified text to the LOG file. 528
Sends specified text to the LOG file. 528, 662
Way to send text to the LOG file. 662
$Macro

© 2022 Prof. Bruce McCarl

$Macro
Dollar command that defines a new macro 528
$Maxcol
Dollar command that sets right margin for the input file. 529
Set right margin for the input file and allows comments 238
$Maxgoto
Maximum number of jumps to the same label 529
$Mincol
Dollar command that sets left margin for the input file. 529
Set left margin for the input file and allows comments 238
$Offdelim
Dollar command that deactivates CSV separation of table data. 529
Turn off GAMS recognition of CSV delimiters. 499
$Offdigit
Dollar command that deactivates significant digit transformation. 529
$Offdollar
Dollar command that suppresses echo print of dollar commands in LST file. 530
Stops echo print of dollar command options in LST file. 137
$Offdotlp
End automalit .l addition 530
$Offecho
Dollar command to stop action of $onecho. 531, 662
$Offembedded
Dollarcommand to disable use of embedded values in parameter and set data statements 532
$Offempty
Dollar command that prohibits empty data statements. 532
$Offend
Dollar command that deactivates alternative syntax for flow control statements. 533
Dollar command which deactivates alternative syntax for flow control statements. 334
$Offeolcom
Deactivate end-of-line comments 236
Dollar command that deactivates ability to use end-of-line comments. 533
$Offeps
Dollar command that deactivates treatment of zeros as EPS. 533
$Offexpand
Dollar command to disable macro expansion 534
$Offglobal
Causes dollar commands in main programs to not be honored in included files. 499
Dollar command that causes dollar commands in main programs to not be honored in included files. 534
$Offinclude
Dollar command that suppresses echo print of included files in LST file. 534
Remove echo print of included files in LST file. 498
Suppresses echo print of included files. 137
$Offinline
Deactivate in line comments 237
Dollar command that deactivates ability to use in line comments. 534
$Offlisting
Dollar command that deactivates echo print of subsequent input lines. 535
Suppress echo print of lines in LST file. 494
Suppresses lines from echo print listing. 137
$Offlocal
Dollar command that limits use of .local on the same symbol to one in one control stack 535
$Offlog
Dollar command that turns off line logging 535
$Offmacro
Dollar command to disable macros 536
$Offmargin
Deactivate margin marking 238
Dollar command that turns off margin marking. 536
$Offmulti
Dollar command that prohibits multiple data item definitions. 537
$Offnestcom
Dollar command that prohibits nested in line comments. 537
Prohibit nested in line comments 237
$offorder
Causes leads and lags to work with unordered sets 538
$Offput
Dollar command stopping put of text block. 539
$Offrecurse

© 2022 Prof. Bruce McCarl

$Offrecurse
Dollar command that disallows a file to include itself 539
$Offstrictsingleton
Turn off error messages for multientry sinlgetons 539
$Offsymlist
Dollar command that removes symbol list from LST file. 540
Removes symbol listing from LST file. 144
$Offsymxref
Dollar command that removes symbol cross reference from LST file. 540
Removes cross reference listing from LST file. 142
$Offtext
Deactivating blocks of code in memory use searches 485
Dollar command that ends a multi line comment. 540
Dollar command that is used to deactivate blocks of code in speed searches. 471
Ends a multi line comment 236
Stop LST file comment. 137
$Offuellist
Dollar command that removes unique element list from LST file. 541
Removes unique element listing from LST file. 145
$Offuelxref
Dollar command that removes unique element cross reference in LST file. 541
Removes unique element cross reference from LST file. 146
$Offundf
Dollar command that prohibits undf from being assigned. 541
$Offverbatim
Dollar command used in conjunction with the GAMS parameter DUMPOPT 541
$Offwarning
Dollar command that activates relaxed domain checking. 542
$Ondelim
Dollar command that activates CSV separation of table data. 529
Turn on GAMS recognition of CSV delimiters. 499
$Ondigit
Dollar command that activates significant digit transformation. 529
$Ondollar
Adds echo print of dollar command options in LST file. 137
Dollar command that adds echo print of dollar commands in LST file. 530
$Ondotl
Automatic .l addition 530
$Onecho
Dollar command to start copying succeeding lines to file. 531, 662
$Onechos
Dollar command that permits parameter substitution when $echo is used 531
$Onechov
Dollar command that forbids parameter substitution when $echo is used 531
$Onembedded
Dollarcommand to enable use of embedded values in parameter and set data statements 532
$Onempty
Dollar command that allows empty data statements. 532
$Onend
Dollar command that activates alternative syntax for flow control statements. 533
Dollar command which activates alternative syntax for flow control statements. 334
$Oneolcom
Activate end-of-line comments 236
Dollar command that activates ability to use end-of-line comments. 533
$Oneps
Dollar command that activates treatment of zeros as EPS. 533
$Onexpand
Dollar command to enable macro expansion 534
$Onglobal
Cause dollar commands in main programs to be honored in included files. 499
Dollar command that causes dollar commands in main programs to be honored in included files. 534
$Oninclude
Begins echo print of included files. 137
Cause echo print of included files. 498
Dollar command that causes echo print of included files. 534

© 2022 Prof. Bruce McCarl

$Oninline
Activate in line comments 237
Dollar command that activates ability to use in line comments. 534
$Onlisting
Activate echo print of lines in LST file. 494
Dollar command that activates echo print of subsequent input lines. 535
Reverses effect of $Offlisting. 137
$Onlocal
Dollar command that allows yunlimited use of .local on the same symbol in one control stack 535
$Onlog
Dollar command that resets line logging 535
$Onmacro
Dollar command to enable macros 536
$Onmargin
Activate margin marking 238
Dollar command that turns on margin marking. 536
$Onmulti
Allow multiple declarations of a named item. 50
Dollar command that allows multiple data item definitions. 537
$Onnestcom
Allow nested in line comments 237
Dollar command that allows nested in line comments. 537
$OnOrder
stops allowing lags on unordered sets 538
$Onput
Dollar command starting put of text block. 539
$Onputs
Dollar command starting put of text block with parameter substitution. 539
$Onputv
Dollar command starting put of text block without parameter substitution. 539
$Onrecurse
Dollar command that allows a file to include itself. 539
$OnStrictSingleton
Turn on error messages for multielement singletons 539
$Onsymlist
Adds symbol listing to LST file. 144
Dollar command that adds symbol list to LST file. 540
Dollar command that adds symbol list to output. 306
$Onsymxref
Adds cross reference listing to LST file. 142
Dollar command that adds symbol cross reference to LST file. 540
$Ontext
Dollar command that starts a multi line comment. 540
Dollar command that stops deactivation of blocks of code in speed searches. 471
Start LST file comment. 137
Start multi line comment 236
Stopping deactivation of blocks of code in memory use searches 485
$Onuellist
Adds unique element listing to LST file. 145
Dollar command that adds UEL list to output. 310
Dollar command that adds unique element list to LST file. 541
$Onuelxref
Adds unique element cross reference to LST file. 146
Dollar command that adds unique element cross reference in LST file. 541
$Onundf
Dollar command that allows undf to be assigned. 541
$Onverbatim
Dollar command used in conjunction with the GAMS parameter DUMPOPT 541
$Onwarning
Dollar command that deactivates relaxed domain checking. 542
$Phantom
Dollar command that designates a phantom set element. 542
$Prefixpath
Dollar commands that augments search path in the windows environment. 543
$Protect
Dollar command that does not allow the objects to be modified in a privacy setting but allows use in model calculations. 543
$Purge
Dollar command that removes the objects and all data associated in a privacy setting. 543
$Remark

© 2022 Prof. Bruce McCarl

$Remark
Dollar command that includes a comment with a substitutable parameter. 544
$Set
Dollar command that defines control variable. 544
Dollar command to set a control variable here and in included code. 629
$Setargs
Dollar command that redefines arguments to a text like name in Batincludes. 544
Dollar command to set arguments for external call. 661
$Setcomps
Dollar command that dissembles period delimited item into individual components. 545
$Setddlist
Dollar command that causes GAMS to look for misspelled or undefined "double dash" – commands. 546
Dollar command to check spelling of - parameters. 659
$Setenv
Dollar command that defines or changes value of environment variable. 547
Dollar command to define or alter value of environment variables 635
$Setglobal
Dollar command that defines global control variable. 546
Dollar command to set a control variable globally. 628
$Setlocal
Dollar command that tears apart a file name into components. 547
Dollar command to set a control variable here. 547, 629
$Setnames
Dollar command that tears apart a file name into components. 548
$Shift
Dollar command that shifts arguments in include files. 548
Shift arguments in include files. 495
$Show
Dollar command that shows control variables. 548
Dollar command to show all control variables and their availability status. 636
$Single
Dollar command that starts single spacing for subsequent echo print lines in LST file. 548
Starts single spaced echo print. 137
$Stars
Dollar command that redefines characters for four **** message. 549
Redefines characters for four **** error messages. 138
$Stitle
Defines subtitle for LST file. 163
Placing a subtitle in a LST file. 549
$Stop
Dollar command that stops compilation. 549
Dollar command to stop a compilation. 663
$Sysinclude
Dollar command that includes file with arguments from system directory. 550
Include file with arguments from system directory. 498
$Terminate
Dollar command to stop a compilation. 550, 663
$Title
Defines LST file title. 163
Placing a title in a LST file. 550
$Unload
Compile time write to GDX file element identification 672
Dollar command that unloads data to a GDX file 550
Problems with compile time write to GDX 672
$Use205
Dollar command that tells GAMS to use version 2.05 syntax. 551
$Use225
Dollar command that tells GAMS to use version 2.25 syntax. 551
$Use999
Dollar command that tells GAMS to use latest version syntax. 551
$Version
Dollar command to test GAMS compiler version number 551

- % -

%
Symbol proceeding batinclude parameter number. 659

© 2022 Prof. Bruce McCarl

%
Symbol that sets off names of control variable, command line parameter or system attribute to be retrieved. 645
%1
Syntax to retrieve batinclude parameter number 1. 659
%Gams.item%
Value of GAMS command line parameter named item. 652
%name%
Syntax to retrieve named control variable, command line parameter or system attribute. 645
%System.item%
Value of windows system environment variable. 654

- (-

( )
Symbol for calculation grouping in equations interchangeable with [ ] and { }. 242

- * -

*
Caution against using in input 169
Symbol for multiplication. 241
Symbol in column 1 rendering line in option file a comment. 812
Symbol that begins a comment, usage to deactivate code in memory use searches 485
Symbol that begins a comment, usage to deactivate code in speed searches. 471
Symbol that when used in set declaration is indicator of universal set. 63
Symbol that will carry out set intersection. 71
Symbol used as to define universal set in report writing. 293
Symbol when in column 1 that begins one-line comments 235
**
Symbol for exponentiation. 241
****
Execution error marker. 441
Marker for compiler error message. 216

- . -

.
Item to set off variable and equation names in specifying MCP complementarity. 87
Symbol to separate set elements when defining multidimensional items 73
..
Symbol signifying start of algebraic specification of a model equation and a dynamic calculation. 240
Symbol signifying start of algebraic specification of a model equation. 83
.. specifications
Algebraic content, tutorial coverage 33
Tutorial coverage 33
..Equation specification
Algebraic specification of a model equation. 83
.Ap
Put file attribute signaling append option. 599
.Bm
Put file attribute specifying bottom margin. 600
.Case
Put file attribute choosing output case control. 604
.Cc
Put file attribute giving current column. 595
.Cheat
Model attribute requiring each new integer solution to be at least a tolerance better than the previous one. 829
.CNS
System attribute usable in a put identifying solver that is currently active for CNS problems. 590
.Cr
Put file attribute giving current row. 596
.Cutoff
Model attribute causing the MIP solver to disregard parts of the tree with an objective worse than a value. 830
.Date
System attribute that identifies date on which model was run. 657
Using date system attribute which identifies date on which model was run in Put files. 590
.Defpoint
Redfining advanced basis 848, 851
.Dirsep

© 2022 Prof. Bruce McCarl

.Dirsep
Attribute giving seperator used by OS in file names 590
.DNLP
System attribute usable in a put identifying solver that is currently active for DNLP problems. 590
.Errors
Put file attribute giving number of put errors encountered. 625
.Fe
System attribute which identifies file extension of input file. 590
.Filesys
System attribute that identifies name of the operating system being used in. 657
.Fn
System attribute giving file name stem of input file. 591
.Fp
System attribute giving file path of input file. 591
.Fx 79
Fixed bounds in calculations. 287
Possible issues when using for MCPs 287, 388
Variable and equation attribute giving solution level. 79
Variable attribute fixing the level to a number. 79
.GamsRelease
System attribute usable in a put identifying GAMS release being used. 657
.GamsVersion
System attribute usable in a put identifying GAMS version being used. 657
.Gstring
System attribute usable in a put identifying specific GAMS version being used. 657
.Handle
Model attribute identifying grid problem 875
.Hdcc
Put file attribute giving current column in header. 596
.Hdcr
Put file attribute giving current row in header. 596
.Hdll
Put file attribute giving header last line. 597
.Ifile
System attribute giving input file name. 591
.Iline
System attribute usable in a put giving number of lines in input file. 591
.Incline
System attribute that identifies line number of include file being executed. 657
.Incname
System attribute that identifies name of file being included. 657
.Incparent
System attribute that identifies parent file that includes this one. 657
.L
Automatic addition of .L 14, 79, 87, 284, 291, 530, 588
Equation attribute giving the solution level or starting point. 87
Tutorial coverage 14
Using solution levels in calculations. 284
Using solution levels in put files. 588
Using solution levels in reports. 291
Variable attribute giving the solution level or starting point. 79
.Lcase
Put file attribute choosing output case control for set elements. 604
.Len
Set attribute giving length of set element name 64
.Lice1
System attribute usable in a put giving GAMS license information. 591
.Lice2
System attribute usable in a put giving GAMS license information. 591
.LicenseStatus
System attribute that identifies if a license problem has arisen 657
.LicenseStatusText
System attribute that returns text describing licensing error 657
.Line
System attribute that identifies line number of overall file being executed. 657
.Lj
Put file attribute specifying Set element name justification. 613
.Ll
Put file attribute giving last line. 597
.Lo
Lower bounds in calculations. 286

© 2022 Prof. Bruce McCarl

.Lo
Variable or equation attribute giving lower limit or bound. 79
.Lp
Put file attribute giving last page. 598
System attribute usable in a put identifying solver that is currently active for LP problems. 591
.Lw
Put file attribute specifying set element name width. 606
.M
Equation attribute giving the solution value for the marginal or starting point. 87
Using solution marginals in calculations. 285
Using solution marginals in put files. 588
Using solution marginals in reports. 291
Variable attribute giving the solution value for the marginal or starting point. 79
.MCP
System attribute usable in a put identifying solver that is currently active for MCP problems. 591
.MINLP
System attribute usable in a put identifying solver that is currently active for MINLP problems. 591
.MIP
System attribute usable in a put identifying solver that is currently active for MIP problems. 591
.Modelstat
Model attribute giving model solution status 854
Putting out numerical model solution status. 587
.MPEC
System attribute usable in a put identifying solver that is currently active for MPEC problems. 591
.Nd
Put file attribute specifying number of decimals. 607
.Nj
Put file attribute specifying numeric field justification. 614
.NLP
System attribute usable in a put identifying solver that is currently active for NLP problems. 591
.NodLim
Model attribute limiting the maximum number of nodes that can be examined in a MIP solution. 830
.Noerror
Extension to $abort 510
.Nr
Put file attribute specifying numeric round option. 619
.Numvarproj
Model attribute giving count of bound projections during model generation 90
.Nw
Put file attribute specifying numeric field width. 607
.Nz
Put file attribute specifying tolerance for when numbers are to be treated as zero. 620
.Off
Set attribute giving ofset position of set elment in ordered set 64
.Ofile
System attribute usable in a put giving output page. 591
.Opage
System attribute usable in a put giving output page. 591
.OptFile 393
Model attribute activating option files for MIP solvers. 832
Model attribute to specify option file presence and relevant file extension. 811
.Ord
Set attribute giving poition of set element in ordered set 64
.Page
System attribute giving current page. 591
.Pc
Put file attribute specifying print control option. 600
.Pdir
Command line parameter specifying where put files will be saved. 578
Put file attribute redirecting the put file output to the scratch directory. 578
.Pfile
System attribute usable in a put giving put file name for currently active file. 591
.Platform
System attribute usable in a put giving computer operating system information. 591
.Pos
Set attribute giving position of set elment in unordered set 64
.Ppage
System attribute usable in a put giving output page 591

© 2022 Prof. Bruce McCarl

.Prior
Variable attribute specifying priority for a variable – the lower the value the higher the priority. 828
.Prioropt
Model attribute activating MIP priorities. 832
.Prline
System attribute that identifies line in output file. 657
.Prpage
System attribute that identifies page in output file. 657
.Ps
Put file attribute specifying page height in lines. 601
.Pw
Put file attribute specifying page width in characters. Max is 32767. 602
.Range
Variable attribute giving difference between upper and lower bounds 79
.Rdate
System attribute giving restart file date. 591
.Rfile
System attribute giving restart file name. 592
.RMINLP
System attribute usable in a put identifying solver that is currently active for RMIP problems. 592
.RMIP
System attribute usable in a put identifying solver that is currently active for RMIP problems. 592
.Rtime
System attribute giving restart file time. 592
.Scale
Calculating variable and equation scaling factors. 288
Variable and equation attribute telling amount to scale that variable or equation. 457
.Scaleopt
Model attribute that activates scaling. 457
.Sfile
System attribute giving save file name. 592
.Sj
Put file attribute specifying set yes no element justification. 615
.Solvestat
Model attribute giving solver termination status 855
Putting out numerical solver termination status. 587
.Sstring
System attribute usable in a put identifying full name of last solver used. 592
.Sw
Put file attribute specifying set yes no element width. 608
.Tab
Placing tabs in put files 602
.Te
Element of put command to use set element explanatory text 233
.Te(setname)
Set attribute giving element explanatory text. 582
.Tf
Put file attribute specifying way to fill missing set element descriptions. 582
.Time
Putting out program execution time. 592
System attribute that identifies time of run. 657
.Title
System attribute giving model title. 592
.Tj
Put file attribute specifying quoted and explanatory text justification. 616
.Tl
Set attribute giving element name. 581
.Tlcc
Put file attribute specifying current column in title block. 598
.Tlcr
Put file attribute specifying current row in title block. 598
.Tlll
Put file attribute specifying last row in title block. 598
.Tm
Put file attribute specifying top margin. 602
.Tmodstat
Model attribute that can be used in put statements giving problem optimality status text 854
Putting out text for model solution status. 587
.TryInt
Model attribute causing MIP solvers to make use of current variable values when solving a MIP problem. 832
.Ts
Identifier containing explanatory text for item. 585
.Tsolstat

© 2022 Prof. Bruce McCarl

.Tsolstat
Model attribute that can be used in put statements giving solver termination status text 855
Putting out text for solver termination status. 587
.Tw
Put file attribute specifying explanatory and quoted text field width. 609
.Uel
Set attribute giving uniquel element list position of set elment 64
.Up
Upper bounds in calculations. 287
Variable or equation attribute giving upper limit or bound. 79
.Val
Set attribute giving numerical counterpart of set element names that are numeric 64
.Version
System attribute that identifies GAMS version number. 657
System attribute usable in a put giving GAMS version being run. 592
.Ws
Put file attribute specifying window size in number of rows. 599

- / -

/
Put file command to skip to new line. 594
Symbol for division. 241
Symbol to set off explicitly defined set elements. 54
Symbol to set off file name definitions. 578
Symbol to set off parameter element definitions 73
Symbol to set off scalar element definitions 72
Symbol used to set of list of equations in a model. 87
/- 659
Command line parameter that allows definition of a control variable. 363

- - -

-/
A sequence initiating definition of a control variable 363, 659
User defined command line parameter. 659

- / -

/ Model contents /
Way of declaring equation presence in a model. 87
//
Command line parameter that allows definition of a control variable. 363
User defined command line parameter. 659

- : -

:
Matching operator for tuples 294, 610, 637
Put file display format delimiter. 610
Symbol that is part of option statement to control decimals and column/row layout. 294

- ; -

;
Symbol that ends statements and when omitted or used excessively is common source of error messages. 221

- ? -

???
Option file name extension when optfile = 100-999. 811

- @ -

@
Gdxxrw command entry alternatives using a text file. 754
Put file command to skip to a column in the put file. 593

- [-

[ ]

© 2022 Prof. Bruce McCarl

[ ]
Symbol for calculation grouping in equations interchangeable with ( ) and { }. 242

- { -

{ }
Symbol for calculation grouping in equations interchangeable with ( ) and [ ]. 242

- + -

+
Augment equations when specifying a model 71, 88
Symbol for addition. 241
Symbol that signifies a lead operation in a set. 69
Symbol that will carry out set union. 71
++
Symbol that signifies a circular lead operation in a set. 70

- < -

<
Put file item left justification symbol. 618
Relation operator in testing whether one item is less than another. 319
Use in inferring set contents from GDX file 319, 525, 556, 618
Use in option command to project items left to right. 556
<=
Defining equation as a less than or equal to. 83
Relation operator in testing whether one item is less than or equal to another. 320
Use in option command to project items right to left. 556
<=>
Relation operator in testing whether one item is logically equivalent to another. 320
<>
Put file item center justification symbol. 618
Relation operator in testing whether one item is not equal to another. 318

- = -

=
Defining equation as an equality relation. 83
Relation operator in testing whether one item is equal to another. 318
Symbol to rename entries in GDX files 673
Symbol used in replacement statements. 240
Symbol used to set items equal to expressions 72
Use in $Call to make GAMS wait for completion of an external program. 693
Use in Execute to make GAMS wait for completion of an external program. 694
==
Symbol used in compile time if test. 645
=b=
Logical equation for EMP model 83
=c=
Symbol identifying equation as a conic equation. 83
=e=
Symbol identifying equation as an equality relation. 83
=g=
Symbol identifying equation as an greater than or equal to. 83
=l=
Symbol identifying equation as an less than or equal to. 83
=n=
Symbol identifying equation as un specified relation. Rarely used but can occur in MCP models. 83
=X=
Equations defined by external programs. 766
Symbol identifying equation as an equation defined by external program. 83

- > -

>
Put file item right justification symbol. 618
Relation operator in testing whether one item is greater than another. 319

© 2022 Prof. Bruce McCarl

- - -

->
Relation operator in testing whether one item logically implies another. 320

- > -

>=
Defining equation as a greater than or equal to. 83
Relation operator in testing whether one item is greater than or equal to another. 319

- 2 -

225a
Temporary GAMS file storage directory 393

- A -

A
Command line parameter that controls the type of compiling action. 363
Abort
Command to display output and stop job. 294
Command to stop GAMS job and display data 316
Conditional job termination and data display. 316
Execution time command that issues an error message in LST file and displays data. 661
Abs 297
Function to find absolute value. 253
Access
Interfacing GAMS with Access. 712
Accessing manuals
Look at solver and other manuals 865
Acronym
Command to assign an item that is a text entry 625
Use of acronyms in calculations. 248
Acronym comparisons
Conditionals over acronyms. 325
Acronyms
Command to assign an item that is a text entry 625
Acrtype
Keyword in compile time $If to see if a named item is an acronym. 648
Action
Command line parameter that controls the type of compiling action. 363
Advanced basis
Advanced basis usage for NLP/MCP model types 842
Ae
Append or overwite expand file 364
Al
Command line parameter that controls the overwriting of the LOG file. 364
Algebra
Tutorial coverage 25
Algebraic
GAMS exploitation of algebraic modeling - tutorial coverage 25
Algorithmic bounds
Adding bounds to improve solver performance 840
Alias
Command giving more than one name to a set. 66
Tutorial coverage 29
All
Key word to include all equations in a model. 87
ALPHAECP
A solver for mixed integer non-linear problems 111
AMPL
Conversion to/from GAMS 785
Using the CONVERT solver to transform a GAMS problem to an AMPL type of problem. 712
And
Operator to form set intersection. 71
Relational operator that links sub-logical conditions being true when all sub conditions are true. 326
ANTIGONE
Solver for global optimization of nonconvex MINLP 111
Ao
Command line parameter that controls the overwriting of the LST file. 364
API to iterface other programs
Interface to GAMS 881
Interface to GDX 881

© 2022 Prof. Bruce McCarl

API to iterface other programs
Interface to Solver options 881
Appendexpand
Allow overwrite or append of expand file 364
Appendlog
Command line parameter that controls the overwriting of the LOG file. 364
Appendout
Command line parameter that controls the overwriting of the LST file. 364
ArcCos(x)
Function giving arc cosine of x 256
ArcSin (x)
Function giving arc sine of x 256
ArcTan(x)
Function to find arctangent. 256
ArcTan2(y,x)
Function that returns four quadrant arctan 256
Arithmetic errors
Errors due to impossible arithmetic operations. 441
Ask
GUI for asking questions of a user 804
Assigned
GAMS concept that data were placed into object by a calculation or solve. 227
Assignment statements
Rules for inclusion of acronyms 626
Attribute
Item for a model referenced by modelname.attribute 846
Solution, bound and scaling factors for a variable. 79
Solution, bound and scaling factors for an equation. 87
Augmentation
Expanding a core model - tutorial coverage 45
Automated problem handling
GAMS capabilities 48

- B -

Baron
Using the CONVERT solver to transform a GAMS problem to a BARON type of problem. 712
BARON solver
A solver for LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP model types that can handle non-convex problems. 112
Conversion to/from GAMS 785
Basis
Advanced basis usage for NLP model types 842
Avoiding problems with bases in comparative analysis. 351
Now obsolete procedure in GAMS to generate an advanced basis and speed up solution. 817
BCH facility
Branch, Cut and Hueristic MIP facility 829
Beta
Beta function. 256
Betareg
Regularized beta function. 256
Binary variable(s)
Declaration of a variable as equal to either zero or one. 77
Binary variables
Variables that can take on values of 0 or 1 only. 823
Binomial
Binomial coefficient 256
Blocklist
GAMSCHK procedure that gives largest and smallest coefficients by variable and equation block. 459
Blockpic
GAMSCHK procedure that gives largest and smallest coefficients by variable and equation block as well as within block intersections. 459
Blue line
Colored navigation line in the process window in the IDE. 183
BONMIN
An experimental MINLP solver 112
BonminD in core link 112
BONMINH a commercial version 112
bool_and
Function that returns 0 if arguments equal zero 1 otherwise 260
bool_eqv
Function that does boolean equivalence 260
bool_imp
Function that does boolean implication 260
bool_not
Function that does boolean not 260
bool_or
Function that does boolean not 260
bool_xor
Function that does boolean xor 260

© 2022 Prof. Bruce McCarl

Branch and Cut Facility
BCH 829
Bratio
Command line parameter 364
Model attribute 551, 557, 818, 847
Option command controlling basis formation. 551, 557
Suppressing or requiring a basis 818
Break
Exiting a loop 334
By
Command in for statement indicating amount to change a scalar varied during each step. 339

- C -

Calendar
Calendar, date and time functions 261
Capitalization
Font case structure that will be used in output. 67
Rules for font case structure that will be used in output. 306
Card
Function that returns the ASCII number for a character in a string. 267
Function that returns total number of elements in a set. 68
Use of function for number of elements in a set in conditionals. 323
Case
Command line parameter that controls the case of text in the LST file for the echo print. 365
CBC
A free open source solver for MIP models 113
CbcD in core link 113
CDF
Cumulative Distribution Funcrion 271
Use in Models 271
Cdim
Total dimension of item in columns in Gdxxrw data specification. 729
Cdir
Command line parameter that gives the name of the current working directory. 366
Ceil
Function to find smallest greater integer. 256
Centropy
Cross entropy function 256
Cerr
Command line parameter that controls compile time error limit. 365
Changing load and unload GDX file names
Using put_utility to change GDX file name 696
Changing put file name
Using put_utility to change put file name 696
Charset
Authorizes extended character set including European and other international characters 232
Command line parameter that allows use of extended character set including European and other international characters 365
Customizing compiler so it includes European and other international characters. 419
cheat
Model attribute that requires a new integer solution to be better 847
Checkver
Checking available updates and license file vintage. 174
CHK4UPD
Checking to see if a newer GAMS is available 882
Cholesky
Utility for Cholesky Decomposition 792
Clear
Clearing old solution option 489, 551, 557, 567, 750
Option command that zeros all data for an item. 551, 557
Option in Gdxxrw. 750
Reducing memory use for an item 489
Click
Put_utility command to put clickable link in process window 696
CNS
Command line parameter that names CNS solver. 366
Constrained nonlinear system model form. 105
Customizable command parameter in Gmsprm file that names CNS solver. 419
Declaration of model type specifying a constrained nonlinear system. 93
Option command that names CNS solver. 551, 557
COINFML
Interfacing COINFML style XML files with GAMS. 712

© 2022 Prof. Bruce McCarl

COINFML
Using the CONVERT solver to transform a GAMS problem to a CoinFML type of problem. 712
Column block
Moving rectangular blocks in middle of lines in the IDE. 195
Comma delimited files
Usage with ondelim/offdelim 499
Command line
Invoking GAMS from the command line. 175
Using command line items in the IDE. 212
Command line entry solver
Altering default solvers through command line 402
Command line GAMS
Tutorial coverage 14
Command line parameters
Alphabetic list of all command line parameters 361
Comments
Why enter 168
Common errors
List of common errors and their cause. 219
Comparative analysis
Analysis over multiple scenarios. 342
Compilation errors
Tutorial coverage 16
Compile errors
Common compilation errors 190, 219
Finding compile errors in the IDE. 190
Complement
Elements that are not in a set. 71
Complementarity
Declaration of complementary relationships. 87
Definition of a complementarity problem. 102
Relationship between variables and equations. 842
Compression
Compress 868
Compressing and encrypting files 868
Conditional
Concept that involves execution of statements only when logical condition is true. 310
Conditionals
Rules for inclusion of acronyms 626
Tutorial coverage 39
Conic equation
Conic equations in GAMS. 83
CONOPT
A solver for CNS, LP, RMIP, NLP, DNLP, and RMINLP model types. 114
CONOPTD
Incore version of CONOPT 114
Console
Putting execution location to console or screen. 352
Constrained nonlinear systems
Constrained nonlinear system model form. 105
Context changes
Changing model domain of applicability -tutorial coverage 44
Continue
Skipping to the end of a Loop 334
Control Structures
For 339
Loop 334
Repeat 317
Skipping or exiting using break or contiinue 336
While 317
Control variable
Variable used in compile time conditional operations. 628
CONVERT
A converter that transforms GAMS models into a format used by other modeling and solution systems. 114
Conversion to/from GAMS 785
Using the convert solver to transform a GAMS problem to a AlphaECP, AMPL, BARON, CoinFML, CplexLP, CplexMPS, Dict, FixedMPS, GAMS Scalar format, LAGO, LGO, LINGO, MINOPT or ViennaDag type of problem. 712
CONVERTD
In core version of CONVERT 115
Cos
Function to find cosine. 256
Cosh
Function to find hyperbolic cosine. 256
Cost ranging
Including output on cost ranging. 156
Courses and Workshops
Lists of 868
CPLEX
Free bare bones version 115, 130
Solver for LP, MIP and RMIP model types. 115
CPLEXD
In core version of CPLEX 115
CPLEXLP
Conversion to/from GAMS 785

© 2022 Prof. Bruce McCarl

CPLEXLP
Using the CONVERT solver to transform a GAMS problem to a CplexLP type of problem. 712
CPLEXMPS 130
Conversion to/from GAMS 785
Using the CONVERT solver to transform a GAMS problem to a CplexMPS type of problem. 785
CPP
cppcclib 275
Function for pdf and cdf gfr normal distributions 275
Cross reference map
Tutorial coverage 17
CSDP
Solver for semidefinite programming 114
CSV files
Passing with GAMS and put. 703
Passing with Rutherford's Gams2csv. 704
Usage with ondelim/offdelim 499
Using to pass information into compiled programs from GAMS. 784
Using to pass information into GAMS from compiled programs. 783
CSV2GDX
Utility to convert CSV file to GDX 796
Curdir
Command line parameter that gives the name of the current working directory. 366
Custom documentation
Providing your own documentation in the IDE. 201
Customize
Procedures to alter GAMS operation on a machine or for a job. 419
cutoff
Model attribute that causes tree to be cut off 847
cvPower
Function that exponentiates an item to a power 256

- D -

Data entry
Tutorial coverage 29
Data reduction
Strategy to zero data to reduce problem size in model debugging. 466
Date
Calendar, date and time functions 261
DB2
Interfacing GAMS with DB2. 712
DEA
A solver for data envelopment and slice problems that uses CPLEX. 116
Decimals
Controlling default decimal places in displays. 297
Option command that controls default decimal places in displays. 551, 558
DECIS
A solver for stochastic linear programs that uses CPLEX. 116
DECISC a version of DESIC that uses CPLEX 116
DECISM a vesion that uses MINOS 116
Declaration
Acronym declaration 625
Definition of models. 82
Definition of variables. 77
Parameter declaration 73
Scalar declaration 72
Table declaration 74
Declared
GAMS concept that item was defined by a scalar, parameter, set, variable, etc. statement. 227
Keyword in compile time $If keyword to see if a named item was declared by a set, parameter etc statement. 650
Decompression
Decompress 868
Decompressing and encrypting files 868
Default bounds on integer variables
Controls default upper bound on integer varibles 559
Defined
Keyword in compile time $If keyword to see if a named item has data. 650
Defining sets from data
Method to compute sets based on data. 65
Degenerate cycling blocking
Improving solver performance avoiding cycling 841
Delphi 781
Example of running GAMS from compiled program 781
Programming language that can run GAMS 781
Descriptive text
Why use 166

© 2022 Prof. Bruce McCarl

Dexist
Keyword in compile time $If to see if a named directory exists. 655
Df
Command line parameter that controls compiler use of alternative date formats 366
Dformat
Command line parameter that controls compiler use of alternative date formats 366
Customizable command parameter in Gmsprm file that controls compiler use of alternative date formats. 419
Diag
Comparing text defining set elements. 69
Function that is a one if text for set elements match and zero otherwise. 323
DICOPT
A program for solving MINLP model types. 116
Non linear mixed integer solver. 833
Dict
Using the CONVERT solver to transform a GAMS problem to a Dict type of problem. 712
dictFile
Function that forces writing a dictionary file 847
Dif1, dif2
Markings that indicates difference in entries in GDX files. 685
Diff
Althernative way to do file comparison 208
Comparing files using the Diff utility 786
Utility to difference two files 786
Diff Textfiles
GAMSIDE feature to difference two files 208
Difference
Elements that differ between sets. 71
Dim
Total dimension of item in Gdxxrw data specification. 729
Dimension
Keyword in compile time $If keyword to see if a named item is of a particular dimension. 650
Discontinous NLP
Discontinuous nonlinear programming model form. 107
Display
Command that causes inclusion of data for an item in LST file. 293
Conditional data displays. 315
Display execution error results in LST file. 441
Tutorial coverage 40
Dispwidth
Changes display width of set elements in columns 558
DIstributed Processing
ALternative to grid computing 870
Using multiple processors 870
Div
Function that ratios two entries 256
Div0
Function like Div but with division by zero protection 256
Dmpopt
List options for a model 559
Dmpsym
Examining memory use by GAMS items 483
Option command that gives data on number of cases stored (memory use) for all GAMS items. 551, 559
DNLP
Command line parameter that names DNLP solver. 367
Customizable command parameter in Gmsprm file that that names DNLP solver. 419
Declaration of model type specifying a discontinuous nonlinear program. 93
Discontinuous nonlinear programming model form. 107
Option command that names DNLP solver. 551, 558
Documentation
Accessing documentation in general. 202
Accessing GAMS documentation in IDE. 196
Supplemental GAMS Corporation materials 868
User generated materials 868
Domain checking
Act of checking if referenced element is in fact in set or if item is defined over named set. 57
Lack of when reading GDX files 673
Domain error
Error when set element is not a member of a set referenced for the position being worked with. 222
Error when set name does not match the set in this position. 223
Domlim
Allowable number of numerical errors in user model nonlinear terms during problem solution 845
Command line parameter that controls the maximum number of domain errors 367

© 2022 Prof. Bruce McCarl

Domlim
Model attribute 558, 845, 847
Option command that specifies maximum number of domain errors. 551, 558
Domusd
Number of numerical errors encountered in user model nonlinear terms during problem solution 845, 851
DOS box
Changing title of during a run. 352
Downto
Command in for statement indicating lower limit for scalar that is decreased as varied. 339
Dp
Command line parameter to dump all include names with paths 370
Dset
Reading sets from lists in Gdxxrw. 740
Dualcheck
Option command that controls dual evaluation in Limcol display. 551, 559
Dumpopt
Keyword that creates a GAMS file of input that will reproduce results encapsulating all include files into one GAMS file 367
Dumpparms
Command line parameter to dump all altered command line parameters 370
Dynamic
Nature of calculations in .. statement. 249
Dynamic set
Calculated set that cannot be used as a domain. 57

- E -

Echo print
Tutorial coverage 15
Economic equilibrium
Tutorial example 5
Edist
Function to compute squareroot of sum of squares of arguments. 256
Ef
Command line parameter that specifies the path name to expand file names with. 374
Eigenvalue
Utility to compute eigenvalues 793
Eigenvector
Utility to compute eigenvectors 794
Eject
Option command that causes a page break in the LST file. 551, 559
Element definition
Act of putting data or elements in a named item. 54
Element name
Use longer names 166
Element order
Order set elements will appear in output. 67
Else
Statement that acts in concert with an If and allows control of multiple lines based on earlier conditionals not being true. 331
Elseif
Statement executed in an If statement context when a condition is true and all previous If and Endif conditions are false. 331
EMP 372
Extended Mathematical Program 107
EMPSP
Stochasic extension of EMP 117
Encryption
Encrypting files 515
Endfor
Alternative syntax which ends for statements under $Onend. 340
Endif
Alternative syntax which ends If statements under $Onend. 334
Endloop
Alternative syntax which ends Loop statements under $Onend. 336
Endogenous function
Indication that GAMS has found nonlinear term in linear model. 226
Endwhile
Alternative syntax which ends While statements under $Onend. 338
Entropy
Function to compute entropy. 256
Environment variables
Gdxcompress and Gdxconvert 633
Windows environment variables. 634
Eolonly
Command line parameter that specifies whether a file defined with pf= has more than one command per line 372
Eps

© 2022 Prof. Bruce McCarl

Eps
Special value depicting a near zero number. 283
Epsout
Writing Eps into spreadsheets with Gdxxrw. 752
Eq
Relation operator in testing whether one item is not equal to another. 318
Eqname.Varname
Syntax used in MCP models to declare complementarity. 87
Equ
Reading and writing equations to/from spreadsheets with Gdxxrw. 746
Equation
Command to declare an equation that can be one of the constraints in a model. 82
Relation within a model that is one of the constraints that must be satisfied in choosing the solution levels. 82
Equation attribute
Solution, bound and scaling factors for an equation. 87
Equation attributes
Use in put files. 588
Equation listing
Algebra use effects on, tutorial coverage 34
Tutorial coverage 18
Equation solution report
Algebra use effects on, tutorial coverage 36
Tutorial coverage 22
Equation table
Assigning values for equation attributes 81
Equations
Algebraic content, tutorial coverage 32
Command to declare an equation that can be one of the constraints in a model. 82
Defined by external programs. 766
Supressing with conditionals 314
Tutorial coverage 9
Equtype
Keyword in compile time $If to see if a named item is an equation. 648
Eqv
Relation operator in testing whether one item is logically equivalent to another. 320
Er
Command line parameter that controls the error messages sent to the LOG file. 373
Errmsg
Command line parameter that controls the position of error messages in the echo print and through use of Errmsg=1 allows one to reposition error messages to just after error marking. 372
Customizable command parameter in Gmsprm file that controls position of error messages in Echo print and through use of Errmsg=1 allows one to reposition error messages to just after error marking. 419
Reposition error messages to just after error marking. 217
Repositions error messages in echo print. 141
Errnam
Command line parameter that specifies name of a file containing error messages. 373
Error
Command line parameter that forces a parameter error with a specified message. 373
IDE facilitation of error discovery. 190
Error message proliferation
Case where one error causes many messages and other than the first one are not really valid. 219
Error repair
Procedures to find and fix compiler errors 219
Errorf
Function to integrate normal distribution. 256
Errorfree
Keyword in compile time $If to see if a compilation has been error free so far. 654
Errorlevel
Function to return completion code of most recent external program called during GAMS run 263
Keyword in compile time $If to execute if less than a specified number of compile errors have occurred. 654
Errorlog
Command line parameter that controls the error messages sent to the LOG file. 373
etAlg
Model attribute - Elapsed time algorithm 851
Etl
Command line parameter that specifies a time limit 373
Etlim
GAMS parameter for specifying a time limit 373
etSolve
Model attribute - Elapsed time solve 851
etSolver

© 2022 Prof. Bruce McCarl

etSolver
Model attribute - Elapsed time solver 851
EXAMINER
A utility that can be used to look at the characteristics of a model solution. 117
Excel
Alternative example of GAMS called from EXCEL 781
Running GAMS from a spreadsheet 769
Exec
Put_utility command to execute extenal program 696
Execerr
Command line parameter that puts a maximum limit on execution errors. 374
Execerror
Function returning number of execution errors. 450
Function to return number of execution errors or reset error count to zero. 263
Way to clear execution error status. 451
Execmode
Command line parameter that controls use of directories and external programs for network administration. 374
Execseed
Function to reset seed or retrieve seed for random number generator. 253
Execute
Changing calling parameters 415, 549, 661, 694
Executing an external procedure at execution time. 694
Executing GAMS from within GAMS 415
Execution time statement that causes a run of an external command or program. 661
Timing of execution. 699
Wait or not for execution to finish 415, 694, 699
Execute =
Use of = in Execute to make GAMS wait for completion of an external program. 694
Execute.ASync 661
Telling GAMS to not wait 694
Execute.AsyncNC
Tells GAMS not to wait and to use another console 696
Execute_Load
Execution time GDX file element reading 675
Variant with domain checking 676
Execute_loaddc
Domain checking at execution time of loaded data 676
Execute_loadhandle
Causes GAMS to load a grid model solution 874
Execute_Loadpoint
Execution time GDX point file element reading 678
Used to load a basis or saved point file 678
Execute_Unload
Execution time GDX file creation 670
Execute_Unloaddi
GDX file with sets items are defined over 670
Executing jobs with substutable strings
Using put_utility to change strings for executing a program 696
Execution error
Error message generated when successfully compiled code is run by GAMS or solver and numerical or other difficulty arises. 439
Execution output
Tutorial coverage 21
Exist
Keyword in compile time $If to see if a named file exists. 655
Exp
Function to find exponentiation of a number. 253
Expand
Command line parameter that specifies the path name to expand file names with. 374
Expandability
Small to large modeling - tutorial coverage 44
Explanatory text
Text which is optional giving explanation of named element or set element 232
External Program
Executing in GAMS. 692
External Programs
Interactively including results during a GAMS run. 761
Extrinsic Function
Froming user defined function 268
Notes on controlling derivatives 839
Ey
Command line parameter that specifies whether a file defined with pf= has more than one command per line 372

© 2022 Prof. Bruce McCarl

- F -

Fact
Function to calculate factorial. 256
Fddelta
Parameter to control numerical gradiant step size 375
Fdopt
Paramter to influcence numerical hessian calculation 376
Ferr
Command line parameter that specifies name and existence of file of compilation error messages. 377
File
Command to define put file names. 578
File comparison 71, 786
Comparing files in the GAMSIDE 208
File not found
Difficulty with IDE project file locations. 213
Filecase
Command line parameter that controls file casing of GAMS generated files. 377
Fileclose
Scripting command 210
Filecompile
Scripting command 210
Fileopen
Scripting command 210
Filerun
Scripting command 210
Files
Command to define put file names. 578
Filesave
Scripting command 210
Filesaveall
Scripting command 210
Filestem
Renaming LST, LOG and LXI files 377
Filewait
Scripting command 210
Filter
Using GDXXRW to put in an Excel filter 751
Filtype
Keyword in compile time $If to see if a named item is a local name for a put file. 648
Find
Finding text in IDE. 192
Find in files
Finding text strings in a group of files with IDE. 193
Fixedmps
Using the CONVERT solver to transform a GAMS problem to a FixedMPS type of problem. 712
Floor
Function to find largest smaller number. 256
For
Executes block of statements for each value of a scalar incremented over a range. 339
Going to bottom or stopping 336, 339
Forcework
Command line parameter that forces workfile translation 378
Forlim
Command line parameter that specifies the maximum number of control structures involving a For, While or Repeat 378
Limits maximum number of passes through For, While and Repeat statements. 339
Option command that specifies maximum number of passes through For, While and Repeat statements. 551, 559
Format
Improving readability of GAMS files 165
Frac
Function to find fractional part of an argument. 256
Free variable(s)
Command declaring variable as one with no restriction. 77
Fsave
Command line parameter that forces GAMS to write a save work file. 378
Function
Command to declare a user defined function 268
User defined functions 268
Function evaluation errors
Numerical errors in user model nonlinear terms during problem solution 845
Function that tests whether an iten is not = to another 260
Funtype
Keyword in compile time $If to see if a named item is a function. 648
Fw
Command line parameter that forces workfile translation 378

© 2022 Prof. Bruce McCarl

- G -

G205
Command line parameter that controls reversions to older versions. 378
Gamma
Gamma function. 256
Gammareg
Regularized gamma function. 256
GAMS
Calling from compiled programs 785
Calling from spreadsheets 769
Executing GAMS from within GAMS. 764
GAMS model library 858
Installation documents 864
Latest version 865
GAMS documentation
Accessing GAMS documentation in the IDE. 198
GAMS FAQ
Frequently asked questions web site 865
GAMS IDE Help
Documentation on IDE accessible through Help. 196
GAMS option statement
List of all option commands 555
Gams2csv
Rutherford's libinclude file to pass CSV data. 704
Gams2tbl
Rutherford utility for output table creation 301
GAMSBAS
Old procedure to generate a Basis 820
Using GDXDUMP to make a basis 798
GAMSCHK
A program designed to aid users examine empirical GAMS models for possible flaws. 117
Procedure that is a GAMS solver that allows one to get information on scaling. 459
GAMSIDE approach 175
Running GAMS Jobs 15
Tutorial coverage 15
GAMS-List
User mailing list 867
Gamsmap
Creatting map output 711
GamsRelease
Function thar returns GAMS release number 263
GAMSsm
Web server that runs GAMS scenarios 785
GamsVersion
Function that returns GAMS version number 263
GAMSworld Google Group
Website with topical area related GAMS discussesions 867
GAMS-X
Web server that runs GAMS 785
Gday
Day of month that corresponds to date. 261
GDir
Command line parameter that specifies the grid file directory 380
Grid computing storage location 880
Gdow
Day of week that corresponds to date (1=Monday,2=Tuesday,..). 261
Gdx
API 881
Backward compatability - gdxcopy 379, 667, 668, 669, 680, 681, 689, 711
Command line parameter that gives the name of and forces writing of GAMS data exchange file. 379
Creating GDX files with command line parameter 668
GAMS data exchange file 667
Interfacing 263, 379, 667, 668, 669, 680, 681, 689, 691, 711, 795
Interfacing with from other programs 263, 379, 667, 668, 669, 680, 681, 689, 691, 711, 795
Reorder viewing of GDX files in the IDE 681
Selected item GDX file 669
Utilities 379, 667, 668, 669, 680, 681, 689, 691, 711, 795
Viewing contents with $Load 379, 667, 668, 669, 680, 681, 711
Viewing GDX files in the IDE 681
Whole problem GDX file 668
Writing GDX data into GMS files 263, 379, 667, 668, 669, 680, 681, 689, 691, 711, 795, 881
Writing older versions 379, 667, 668, 669, 680, 681, 689, 691, 711
Gdx file
Creating a GDX file in GAMS 668
GDX file viewing

© 2022 Prof. Bruce McCarl

GDX file viewing
Examining GDX files in IDE. 681
GDX files
Passing information from other programs. 705
Passing information to other GAMS programs. 764
Using Gdxmerge to compare GDX files 687
GDX point file
Creating a GDX solution point file in GAMS 669
Saved basis in GDX format 819
Gdx2access 797
Utility to pass data to the access database 711
Gdx2har
Utility to convert GDX data to GEMPACK header 689
Gdx2sqlite
Utility to pass data to the SQLITE database 797
Gdx2xls
Utility to put all data in GDX file into Excel 802
Gdxcompress
Writing compressed GDX files 379
Gdxconvert
Writing older GDX file versions 379
Gdxcopy
GDX file backward compatability 689
Gdxdiff
Utility to compare contents differences in two GDX files 685
Gdxdump
Utility to write out contents of GDX file in GAMS format 682
Writing a CSV file 798
Writing a GAMSBAS basis file 682
Gdxin
Put_utility command to change active GDX loading file 696
Gdxmerge
Utility to merge GDX files and compare data items 687
GDXMRW
Matlab interface 711
Gdxout
Put_utility command to change active GDX unload file 696
Gdxrank
Sorting one dimensional items 799
Gdxrename
Utility to rename set elements in a GDX file 800
Gdxrrw
Interface ro the R language 801
Gdxuels
Controls UELS written to GDX files 380
Gdxviewer
Program to link to Excel, Access, CSV, or text files and to plot data. 711
Gdxxrw 777
Command entry alternatives. 754
Command entry using a range in a spreadsheet. 755
Command entry using a text file. 754
Debugging. 759
Errors due to open workbooks. 749
Gdxxrw commands. 754
Ignoring rows and columns in a spreadsheet 726, 727, 729, 730, 743, 746, 749, 752, 754, 755, 757, 759, 777
Log file. 757
Range specification. 727
Read and write Excel spreadsheet data using GDX files. 726
Reading and writing equations. 746
Reading and writing parameters. 743
Reading and writing sets. 731
Reading and writing variables. 746
Sharing workbooks. 749
Skipping empty rows and columns 746
Specification of type of data to read or write with spreadsheets. 729
Tracing performance. 757
Using to pass information to Excel 777
Using to read information from Excel 777
Writing Hyperlinks 730
Writing special values and zeros. 752
Writing text 726, 727, 729, 730, 731, 743, 746, 749, 752, 754, 755, 757, 759, 777
Ge
Relation operator in testing whether one item is greater than or equal to another. 319
GEMPACK
Utilities to convert header array files 689
Generate
Phase of GAMS execution where problem is assembled for transfer to solver. 95
Generating
Phase of GAMS execution where problem is assembled for transfer to solver. 95
Generation listing
Tutorial coverage 18

© 2022 Prof. Bruce McCarl

Geographic mapping
Gamsmap 711
Mapping GAMS Output 711
Shademap 711
Ghour
Hour of day that corresponds to date. 261
GLB file 263
Defining a custom user library 862
Gleap
Indicator of whether the year that corresponds to date is a leap year (0=no leap year, 1=leap year). 261
Global
Type of control variable defined everywhere in code. 628
GLOMIQO
Global Mixed-Integer Quadratic Optimizer 118
Gmillisec
Milliseconds that corresponds to date. 261
Gminute
Minute that corresponds to date. 261
Gmonth
Month that corresponds to date. 261
GMS file
Default GAMS file extension. 178
GMS processor
Making IDE the program called when GMS file is clicked on. 181
Gms2tabl
Use of Ruterfords Gms2tabl to write HTML and LATEX 712
Gmsprm98.txt
File that can be used to customize GAMS function on a windows 95/98 machine. 420
Gmsprmnt.txt
File that can be used to customize GAMS function on a Windows NT machine. 420
Gmsprmun.txt
File that can be used to customize GAMS function on a Unix or Linux machine. 420
GMSUNZIP
Utility to UNZIP files 809
GMSZIP
Utility to form ZIP files 809
Gnuplot
Procedures to construct graph of GAMS data. 302
Use in GAMS to graph. 706
Use of Rutherford's libinclude to graph. 706
Gnupltxy
Procedures to construct graph of GAMS data. 302
Use of Schneider and McCarl's libinclude to graph. 706
Good modeling practices
Tutorial coverage 48
GPR file
IDE project file. 177
Gradient
First derivative of nonlinear coefficients at current or starting point that is used in model solution. 459
Graphics
Entering statements into a GAMS program that permit graphical displays. 706
Graphing
Procedures to construct graph of GAMS data. 302
Grid Computing
Distributed processing alternative 380, 870, 874, 875, 878, 879, 880
File storage 870, 874, 875, 877, 878, 879, 880
Functions used 870, 878
General use 870, 874, 875, 877, 878, 879
Gridscript 380, 870, 874, 875, 877, 878, 879, 880
Handle definition and use 870, 874, 875, 878, 879
Invoking 870, 874, 878, 879
Load from GDX 870, 874, 875, 877, 878, 879
Solution inclusion in LST file 870, 877
Solution Retrieval 870, 874, 875, 877, 878, 879
Griddir
Command line parameter that specifies the grid file directory 380
Grid computing directory 880
Gridscript
Command line parameter that provides the name of a script file to submit grid computing jobs 380
Gscript
Command line parameter that provides the name of a script file to submit grid computing jobs 380
Gsecond
Second that corresponds to date. 261
Gt
Relation operator in testing whether one item is greater than another. 319
GUROBI

© 2022 Prof. Bruce McCarl

GUROBI
A LP and MIP solver 118
Free bare bones version 118, 130
GUSS
Solver for repeated scenarios 118
Gyear
Year that corresponds to date. 261

- H -

Handle
Identifies problems in grid computing 875
Model attribute - Every solve gets a unique handle number 851
HandleCollect
Function that retrieves Grid Computing solutions 263
HandleDelete
Function that deletes Grid Computing problems 263
HandleStatus
Constants 878
Function that retrieves Grid solutions into GDX file 263
Status of model solve in Grid computing 263, 874
HandleSubmit
Function that resubmits Grid Computing problems 263
Har2gdx
Utility to convert GEMPACK header array to GDX 689
heapFree
Function giving allocated memory which is not in use but is not freed yet 263
HeapLimit 252
Command line parameter for limiting GAMS memory use 381
Function to control memory use 263
Heapsize
Function to recover the heap size in million bytes. 263
Hierarchy
Hierarchy of GAMS customization procedures. 423
Hl
Command line parameter for limitting Gams memeory use 381
holdFixed
Command line parameter that controls treatment of fixed variables as constants 381
Possible issues with MCP complementarity 102, 381
Treat fixed variables as constants 847
HTML
Interfacing GAMS with HTML. 712
Writing items in GDX file to HTML in the IDE 681

- I -

I
Command line parameter giving the input file name 382
ICDF
Inverse Cumulative Probability Distribution Function 271
Use in Models 271
Icon
Making IDE icon. 214
IDE
Command line parameter controlling output of special instructions to the log file 381
GAMS integrated development environment program that allows editing and execution. 175
Putting execution location to IDE process window. 352
Viewing GDX files in the IDE 681
IDE documentation
Accessing documentation in the IDE. 197
Discussion of IDE features. 175
IDE refreader
Use of refreader program for unraveling complex files in IDE. 202
Idir
Change where included and Batincluded files come from. 494
Command line parameter which gives directory where included and batincluded files are kept. Can include multiple directories. 382
Idir1 to Idir40
Command line parameters that gives input search path where included and batincluded files are kept. 383
If
Conditional control of multiple lines. 316
Statement that allows control of multiple lines based on conditional. 331
Ifthen

© 2022 Prof. Bruce McCarl

Ifthen
Function setting a value to one of two expressions depending on a conditional. 254
IgnoreColumns
Ignoring columns when reading a spreadsheet 746
IgnoreRows
Ignoring rows when reading a spreadsheet 746
Imp
Relation operator in testing whether one item logically implies another. 320
Inc
Put_utility command to include file in put file 696
Include data
Including data from other programs. 713
Indenting
Improving readability of GAMS files 171
Index
Gdxxrw command entry alternatives using a range in a spreadsheet. 755
Inf
Depicts number as infinity in assignment statement. 297
Special value depicting infinity can be used in replacement statement. 283
-Inf
Special value depicting negative infinity can be used in replacement statement. 283
Initial values
Starting values provided for the decision variables within the problem 838
Initialized
GAMS concept that data were placed into object when type was declared (by scalar, parameter ... statement). 227
Input
Command line parameter givin the input file name 382
Inputdir
Command line parameter that gives input search paths where included and batincluded files are kept. Can include multiple directories. 382
Customizable command parameter in Gmsprm file that gives input search paths. Can include several search paths separated by OS specific symbols. 419
Inputdir1 to 40
Command line parameters that give input search path. 383
Customizable command parameter in Gmsprm file that gives input search path names to be used. Default is no search path. 419
Ins1, ins2
Markings that indicates inserts or deletions in GDX files. 685
Installation
GAMS installation documents 864
Integer variable(s)
Command declaring variable as equal to a non-negative integer. 77
Integer variable bounds 77, 835
Integer variables
Variables that must take on integer values 823
integer1-5
Integer communication cell 383, 847
Interface other programs with GAMS
API to interface with GAMS 881
API to interface with Solver options 881
API to read/write GDX files 881
Intersection
Method to define common set elements. 71
IntVarUp
Contorls bounds on integer variable 383, 559
Setting in an option statement 383
Stting in command line 559
Invert
Matrix inversion 791
IPOPT
A free open source interior point solver for NLP models. 126
CoinIpoptD an in core link 126
IPOPTH a commercial version 126
Item name
Names for sets, scalars, put files, parameters, tables, acronyms, variables, equations and models 232
Use longer names 166
Item order
Rearranging in Gdxxrw. 749
Iteration log
Influence of NLPs 844
Iterlim
Command line parameter that specifies the maximum number of allowable solver iterations 384, 440, 551, 560
Expand maximum number of solver iterations. 440
Model attribute 440, 560, 847

© 2022 Prof. Bruce McCarl

Iterlim
Option command that specifies maximum number of solver iterations. 551, 560
iterUsd
Model attribute - Number of iterations used 851

- J -

JAMS
Solver for EMP problems 126
Java
Web based Programming language that can run GAMS 785
Jdate
Gregorian date corresponding to year, month and day. 261
Jnow
Current time. 261
jobHandle
Function that returns the Process ID of last job started 263
jobKill
Function that sends a kill signal to the runnuing job 256
jobStatus
Function that status of a job 263
jobTerminate
Function that kills a job 263
Jobtrace
Command line parameter that specifies a string written to the trace file 384
Jstart
Time of the start of the GAMS job. 261
Jt
Command line parameter that specifies a string written to the trace file 384
Jtime
Fraction of a day that corresponds to this hour, minute and second. 261

- K -

keep
Command line paramater that tells GAMS to keep temporary files 385
KESTREL 127
Web server that runs GAMS 785
Kill
Option command that removes all data for an item. This should not usually be used. Use Clear instead. 551, 560
Removing memory use for an item 489
KNITRO
A solver for NLP model types 127

- L -

Lago
Using the CONVERT solver to transform a GAMS problem to a LAGO type of problem. 712
Large model facilities
Tutorial coverage 47
Latex
Interfacing GAMS with Latex. 712
Ldir
Change where Libincluded files come from. 498
Command line parameter that gives directory for libincluded files. 385
Le
Relation operator in testing whether one item is less than or equal to another. 320
Level
Solution value for variable 42, 118
Use in GUSS 118
Lf
Command line parameter that gives name of the LOG file. 386
LGO
A solver for LP, NLP, DNLP, RMINLP, and RMIP model types that can handle non-convex problems. 127
Conversion to/from GAMS 785
LGOD an in core link 127, 785
Using the CONVERT solver to transform a GAMS problem to a LGO type of problem. 712
Libincdir
Command line parameter that gives directory where libincluded files are kept. 385
Customizable command parameter in Gmsprm file which gives directory for libincluded files. 419
Library
Defining your own library 48, 178, 861
GAMS Data Utilities Library 862
GAMS EMP Library 862
GAMS Financial Library 862
GAMS model library 858

© 2022 Prof. Bruce McCarl

Library
GAMS or user defined collection of GMS and other files accessed through IDE. 178
GAMS Test Library 48, 178, 858, 861, 862
Tutorial coverage 48
License
Command line parameter that gives name of the GAMS license file. 385
Gaining access to solvers. 108
General comments 882
License file
Checking available updates and license file vintage with Checkver. 174
Maintaining license files on computer. 174
LicenseLevel
Function that returns indicator of license error 263
LicenseStatus
Function that returns non zero under a license error 263
Licensing
Gaining access to solvers. 108
Limcol
Command line parameter that includes the first n cases for each named variable in the LST file. 385
Control length of variable print out in output. 163
Customizable command parameter in Gmsprm file that that controls number of variables printed out under each variable block. 419
Model attribute 847
Nonlinear item marking 151, 163, 385, 419, 458, 551, 560, 843
Option command that that controls number of variables printed out under each variable block. 551, 560
Tool to examine scaling. 458
Variable print out in output. 151
Limrow
Command line parameter that includes the first n cases for each named equation in the LST file. 386
Control length of equation print out in output. 163
Customizable command parameter in Gmsprm file that controls number of equations printed out in each equation block. 419
Equation print out in output. 149
Model Attribute 847
Nonlinear item marking 149, 163, 386, 419, 458, 551, 560
Option command that controls number of equations printed out in each equation block. 551, 560
Tool to examine scaling. 458
LINDO/LINDOGLOBAL
LINDOGLOBAL solver for MINLP problems 128
line
Model attribute - Line number of last solve 851
Linear programs
Linear programming model form. 99
LINGO
Conversion to/from GAMS 785
Using the CONVERT solver to transform a GAMS problem to a LINGO type of problem. 712
linkUsed
Model attribute - SolveLink used for the last solve 851
Ll
Command line parameter that controls line tracing 386
Lo
Command line parameter destination for the LOG file, used with setting of 0 or 2 to permit Unix/Linux jobs to operate in background. 387
Local
Type of control variable defined locally in code. 629
Log
Adding message to LOG file 254, 352, 579, 696, 757
Directing put output to log file. 579
Function to find logarithm base e of a number. 254
Gdxxrw Log. 757
Putting execution location to log file. 352
LOG file
File created when GAMS runs which is echoed to the screen and IDE process window. 134
Log10
Function to find logarithm base 10 of a number. 254
Log2
Function to find logarithm base 2 of a number. 254
Logappend
Appending to Gdxxrw Log. 757
Logbeta
Log of beta function. 256

© 2022 Prof. Bruce McCarl

Logfile
Command line parameter that gives name of the LOG file. 386
Renaming with Filestem 377, 386
Loggamma
Log gamma beta function. 256
Logline
Command line parameter that controls line tracing 386
Logoption
Command line parameter controls destination for the LOG file, used with setting of 0 or 2 to permit UNIX jobs to operate in background. 387
Customizable command parameter in Gmsprm file that controls destination for the Log file, used with setting of 0 or 2 to permit Unix jobs to operate in background. 419
Loop
Executes block of statements for each element of a set. 334
Exiting early with Break or Continue 336
Use in comparative analysis. 348
LP
Command line parameter that names LP solver. 387
Customizable command parameter in Gmsprm file that names LP solver. 419
Declaration of model type specifying a linear program. 93
Linear programming model form. 99
Option command that names LP solver. 551, 561
LST file
Main output file for a GAMS run. 134
Navigation Window 185
Renaming with Filestem 134, 185, 377
Lt 185
Relation operator in testing whether one item is less than another. 319
LXI file
Navigating LST using LXI file 185
Renaming with Filestem 377

- M -

Macro
Multi-line macros 571
Running GAMS through predefined script 569
Macros
Showing active macros 548, 569
Using macros to define terms in equations 569
Mapping Programs
Drawing geographic maps from GAMS. 711
Mapval
Function that returns numeric codes for special values. 256
Marginal
Recovering marginals from solution 42
Use in GUSS tuple 118
Marginals
Atrribute telling if solver retruned dual solution 854
Match parentheses
Parentheses matching with IDE. 194
Matching operator :
Defing a tuple with : 62
Matchit
GAMSCHK procedure that gives largest and smallest coefficients by variable and equation. 459
Matlab
Interface with Matlab software. 711
Matrix utilities
Cholesky factorization 790, 791
Eigenvalue 790, 791, 793
Eigenvector 790, 791, 793, 794
Invert 790, 791
Max
Component of model statement indicating model is to be maximized. 93
Function to find maximum among numbers. 254
Maxexecerror
Function to read or reset number of execution errors. 263
Maximizing
Component of model statement indicating model is to be maximized. 93
Maxinfes 851
Solver reported maxinfes 851
Solver reported value of maxinfes 851
Maxprocdir
Maximum number of 225a, 225b etc directories 387
MCfILTER 389
Filter dominated points from GDX file 804
MCP
Command line parameter that names MCP solver. 387

© 2022 Prof. Bruce McCarl

MCP
Complementarity issues 93, 102, 387, 419, 551, 561
Customizable command parameter in Gmsprm file that names MCP solver. 419
Declaration of mixed complementarity problem model type. 93
Mixed complementarity problem model form. 102
Option command that names MCP solver. 551, 561
MCP complementarity
Difficulties and MCPrHoldFx 102, 842
Specifying complementarity. 842
MCP solution output
Differences in output when MCP models are solved 845
MCPrHoldFx
Lists equations complementary with fixed variables 388
Mdb2gms
A program to generate an include file from contents of an Excel spreadsheet. 717
Meaninfes
Solver reported value of meaninfes 851
Measure
Option command that causes output of time and memory since last measure command or program beginning. 551, 561
Memory problems
Investigating excess memory use 478
Merge
Keyword for Solveopt option causing merging of solution information. 567
Option in Gdxxrw. 749
Solveopt option that causes solution to be merged. 97
Message
Scripting command 210
MessageReceiverWindow
Making windows recieve a meassage from GAMS 698
MILES
A solver for MCP model types. 128
MILESE
The newest version of MILES. 129
MILESOLD
A discontinued version of MILES. 129
Min
Component of model statement indicating model is to be minimized. 93
Function to find minimum among numbers. 254
Minfout
Writing -Inf into spreadsheets with Gdxxrw. 753
Minimizing
Component of model statement indicating model is to be minimized. 93
MINLP
An option keyword used to define the currently active MINLP solver. 834
Command line parameter that names MINLP solver. 388
Customizable command parameter in Gmsprm file that names MINLP solver. 419
Declaration of model type specifying a mixed integer nonlinear program. 93
Mixed integer nonlinear programming model form. 103
Option command that names MINLP solver. 551, 561
MINOPT
Conversion to/from GAMS 785
Using the CONVERT solver to transform a GAMS problem to a MINOPTtype of problem. 712
MINOS
A solver for DNLP, NLP and RMINLP model types. 129
MINOSD- an in core verision 129
MINOS5
An older version of MINOS. 129
MIP
An option keyword used to define the currently active MIP solver. 834
Branch, Cut and Hueristic MIP facility 93, 100, 388, 419, 551, 561, 821, 829, 834
Command line parameter that names MIP solver. 388
Customizable command parameter in Gmsprm file that names MIP solver. 419
Declaration of model type specifying a mixed integer program. 93
Discussion of a basis in a MIP context. 821
Mixed integer programming model form. 100
Option command that names MIP solver. 551, 561
MIQCP
Command line parameter that names MIQCP solver 388

© 2022 Prof. Bruce McCarl

MIQCP
Relaxed mixed integer quadratically constrained programming model form. 104
Mismatched parentheses
Common source of errors. 224
Mixed complementarity programming
Mixed complementarity programming model form. 102
Mixed integer NLP programming
Mixed integer nonlinear programming model form. 103
Mixed integer programming
Mixed integer programming model form. 100
Mod
Function to find modulus of a number. 256
Model
Command that groups a number of equations into a named item that can be solved. 87
Tutorial coverage 11
Model attribute
Way of addressing solution status and model options for a particular model. 289
Model attributes
Attribute of a model giving solution performance or specifying procedures 847
Use in put files. 587
Model library 858
GAMS model library 858
GAMS or user defined collection of GMS and other files accessed through IDE. 178
Library of models in this manual 862
Tutorial coverage 48
User defined model library 858
Model readability
Enhancing through formatting 165
Model setup output
Influence of NLPs 844
Model statistics
Tutorial coverage 21
Model type
Form of problem to be solved: LP, NLP, MIP etc. 93
Model types
Alternative problem types that can be solved. 99
Models
Command that groups a number of equations into a named item that can be solved. 87
Modelstat
Indicator of solution status 854
Model attribute indicating the model status 851
Modtype
Keyword in compile time $If to see if a named item is a named model. 648
MOSEK
A solver for LP, MIP, RMIP, NLP, DNLP, and RMINLP model types. 129
Free bare boines version 129
Mp
Command line parameter that causes GAMS to employ a quick syntax check. 389
MPEC
Command line parameter that names MPEC solver. 388
Customizable command parameter in Gmsprm file that names MPEC solver. 419
Declaration of model type specifying a mathematical program with equilibrium constraints. 93
MPEC models
Mathematical program with equilibrium constraints model form. 106
MPS
Interfacing GAMS with MPS based solvers. 712
MPSGE
A preprocessor that aids in formulation and solution of general equilibrium models. 129
Msappavail
Checks for presence of Microsoft office software 807
Msg
Put_utility command to put message in LST file 696
Msglog
Put_utility command to add message to LOG anf LST files 696
MSNLP
A solver for NLP, DNLP, RMINLP, and MINLP model types that can handle non-convex problems. 130
Multi dimensional set
Set that is defined with respect to more that one other set. 56
Multi_threading
Including solutions in LST file 400
Invoking using Solvelink=6 400
Multipass
Command line parameter that that causes GAMS to only do a quick syntax check. 389
Multiple solve

© 2022 Prof. Bruce McCarl

Multiple solve
Procedure containing more than one execution of solve statement. 96

- N -

Na
Depicting a number as unavailable in assignment statement. 297
Special value depicting an unavailable number, can be used in assignment statement. 283
NameConv
Changing range specification convention in Gdxxrw 728
Naout
Writing Na into spreadsheets with Gdxxrw. 752
Navigate
Aids to access LST and GMS file in IDE. 183
Nc
Changing range specification convention in Gdxxrw 728
Ncpcm
Function for use in smoothing functions in MPEC models with Chen-Mangesarian approach. 256
Ncpf
Function for use in smoothing functions in MPEC models with Fisher approach. 256
ncpVUpow
Function for NCP Veelken-Ulbrich smoothed min 256
ncpVUsin
Alternative function for NCP Veelken-Ulbrich minimum 256
Ne
Relation operator in testing whether one item is not equal to another. 318
Negative variable(s)
Command declaring variable as non-positive. 77
Nested conditionals
Use of multiple layers of conditionals. 326
Newsletter
Bruce McCarl's newsletter 867
NLP
Command line parameter that that names NLP solver. 389
Computing derivatives 93, 99, 389, 551, 562, 822, 839
Declaration of model type specifying a non linear program. 93
Discussion of a basis in a NLP context. 822
Nonlinear programming problem form. 99
Option command that names NLP solver. 551, 562
NLP and MCP variants
Nonlinear and mixed complementarity problem forms 846
NLPEC
A solver for MPEC models. 130
No
Removes a set element through a calculation. 284
Special value that indicates an element is not in a set, can be used in assignment statement. 56
nodLim
Command line parameter that controls the maximum number of nodes in a branch an bound tree 389
Model attribute for maximum number of branch and bound nodes 847
nodUsd
Model attribute - Number of nodes used by MIP solver 851
Nonewvarequ
Prohibits adding variable and equations 389
Nonlinear equation system
Tutorial example 6
Nonlinear MIPs
Integer programming problems with nonlinear terms. 833
Nonlinear program
Nonlinear programming model form. 99
NonNegative Variable
Specifies variable as greater than or equal to zero 77
Normal
Bivariate distribution 275
Function to generate a random normal number. 256
PDF and CDF generation 256, 275
Trivariate distribution 275
Not
Operator to form set complement. 71
Relation operator that makes a condition true what a logical condition is false. 328
Relation operator that makes a condition true when a $If logical condition is false. 639
number
Model attribute containing a serial model number 851
NumCores

© 2022 Prof. Bruce McCarl

NumCores
Number of logical cores on computer 267
numDepnd
Model attribute - Number of dependencies in a CNS model 851
numDVar
Model attribute - Number of discrete variables 851
numEqu
Model attribute - Number of equations 851
numInfes
Model attribute - Number of infeasibilities 851
numNLIns
Model attribute - Number of nonlinear instructions 851
numNLNZ
Model attribute - Number of nonlinear nonzeros 851
numNOpt
Model attribute - Number of nonlinear nonzeros 851
numNZ
Model attribute - Number of nonzero entries in the model coefficient matrix 851
numRedef
Model attribute - Number of MCP redefinitions 851
numVar
Model attribute - Number of variables 851
numVarProj
Model attribute - Number of bound projections during model generation 851

- O -

O
Command line parameter that gives the name of the file containing the output. 391
O??
Option file name extension when optfile = 10-99. 811
objEst
Model Attribute - Estimate of the best possible solution for a mixed-integer model 851
objVal
Model attribute - Objective function value 851
Offput
Dollar command stopping put of text block. 621
On115
Command line parameter tha generates errors for unknown unique element in an equation 390
Onput
Command starting put of text block. 621
Onputs
Command starting put of text block with parameter substitution. 621
Onputv
Command starting put of text block without parameter substitution. 621
Op?
Option file name extension when optfile = 2-9. 811
Opt
Default extension for option file names when optfile=1. 811
Optca
Comamnd line parameter that specifies an absolute termination tolerance 390
Model attribute or option command telling the solver to stop when best solution is no more than a given amount away from the best solution. 831, 847
Option command that specifies absolute optimality tolerance in a MIP. 551, 562
Optcr 831
Command line parameter that specifies a relative optimality criterion 390
Default value. 831
Model attribute or option command telling the solver to stop when best solution is no more than a given proportion of the best solution away from the best solution. 831, 847
Option command that specifies relative optimality tolerance in a MIP. 551, 562
Optdir
Command line parameter that gives the name of the directory to be used by GAMS for solver option files. 390
Command line parameter to define a central location for option files. 816
Optfile
Command line parameter that gives the number to use for model.optfile. 391
Command line parameter to specify option file presence and relevant file extension. 816
Customizable command parameter in Gmsprm file that specifies default value for model.Optfile, can be set to 1 if one always wants GAMS to look for option file. 419
Model attribute 391, 419, 816, 847

© 2022 Prof. Bruce McCarl

Optfile =0
Optfile setting that results in no option file used. 811
Optfile > 1
Option file setting that renders file extension op2-op9, o10-o99, or 100-999. 811
Optfile=1
Optfile setting that results in option file "Opt" being used. 811
Optimization problem
Tutorial example 5
Option
Command that invokes GAMS execution options. 552
List all options 552, 559
Option <
Project items left to right. 556
Option <=
Project items right to left. 556
Option file
Option file editor 810, 813, 815
Solver Option files 810, 815
Writing in job. 815
Option itemname
Command to format display appearance. 294
Option itemname:d
Option command controlling display item formatting. 555
Option itemname:d:r:c
Option command controlling display item formatting. 555
Options
Command that invokes GAMS execution options. 552
Or
Relational operator that links sub-logical conditions being true when at least one sub condition is true. 327
ORACLE
Database that can control GAMS runs 785
Interfacing GAMS with Oracle. 712
Ord
Function that returns number of element in set. 323
Function that returns position number of set element in total set. 67
Function that returns the ASCII code numbers for a character in a string. 266
Probems with unordered sets 67, 230, 266, 323
Ordascii
Function that returns the ASCII code numbers for a character in a string. 266
Ordebcdic
Function that returns the EPCDIC code numbers for a character in a string. 266
Order
Order of set elements as they will appear in the output. 67
Ordering of set elements and item names in output. 307
Ordered set
Problems with ORD and unordered sets 67, 230
Set that has ordered elements and can be used with Ord, leads and lags. 67
Ordering
Ordering of set elements and item names in output. 307
OSICPLEX
Free bare bones version of CPLEX 130
OSIGUROBI
Free bare bones verion of GUROBI 130
OSIMOSEK
Free version of MOSEK 131
OSISOPLEX
Free LP solver 131
OSIXPRESS
Free bare bones version of XPRESS 131
Output
Command line parameter that gives the name of the file containing the output. 391
Output navigation
Navigating LST file using LXI file 185
Output to other programs
Procedures to send data to other programs. 302

- P -

Page height
IDE page height setting. 180
Page width
IDE page width setting. 180
Pagecontr
Command line parameter that tells the default put file page control to use. 391
Pagesize

© 2022 Prof. Bruce McCarl

Pagesize
Command line parameter that tells the default number of lines per page. If less than 30 it will be reset to the default of 9999. 391
Pagewidth
Command line parameter that tells the default number of columns on a page. 392
Par
Reading and writing parameters to/from spreadsheets with Gdxxrw. 743
Param
Use in GUSS tuple 118
Parameter
Command to define a data item 73
Parameters
Command to define a data item 73
Tutorial coverage 30
Parentheses match
Parentheses matching with IDE. 194
Parmfile
Command line parameter that gives the name of the GAMS supplemental command line parameter file to use. 392
Partype
Keyword in compile time $If to see if a named item is a parameter. 648
PATH
A MCP, CNS, and NLP (through PATHNLP) solver. 131
PATHC
The latest version of PATH. 132
PATHNLP
A variant of PATH that can solve LP, NLP, RMIP and RMINLP model types. 132
PATHOLD
A discontinued, older version of PATH. 132
Pc
Command line parameter that tells the default put file page control to use. 391
PDF
Use in models 271
Pdir
Command line parameter that gives the name of the directory where files generated by the Put command will be stored. 395
Customizable command parameter in Gmsprm file that identifies put directory. If not specified, it will be set to the work directory. 419
Percentage change
Computing percentage changes. 346
Pf
Command line parameter that gives the name of the GAMS supplemental command line parameter file to use. 392
Customizable command parameter giving a name of a file that contains command line parameters. 420
pf4
Abbreviation for Intvarup 835
Pi
Function to deliver value of pi-3.141716…. 256
Picture
GAMSCHK procedure that gives indication of coefficient magnitude for variables and equation plus reveals structure. 459
Pinfout
Writing +Inf into spreadsheets with Gdxxrw. 753
Plicense
Privacy license command line parameter 393
Poly
Polynomail expansion function 256
Portability
Platform independence 48
Positive variable(s)
Command declaring variable as non-negative. 77
Posix
File maipulation utilities 786
Power
Function to exponentiate a number. 256
Precedence order
Precedence order incorporating numbers and logical conditions. 329
Prefix for dumped command line parameters 371
Presolve
Problem preprocessing done by solvers. 447
Priorities
Way of specifying an order for picking variables to branch on during a MIP branch and bound solution. 828
priorOpt
Model attribute on MIP priorities 847
Probability functions
PDF, CDF, ICDF 268, 271
Random Numbers From 275
Use in model equations 268, 271
ProcDir
Way of changing temporary file name 393
Process window

© 2022 Prof. Bruce McCarl

Process window
IDE version of LOG file. 181
Putting execution location to IDE process window. 352
procUsed
Model attribute - Used model type 851
Prod
Function to multiply elements over a set. 254
Profile
Command line parameter that causes GAMS to include information on statement execution time and memory use in LST file allowing one to find slow or large memory using statements. 393
Customizable command parameter in Gmsprm file that causes GAMS to include information on statement execution time and memory use in Lst file allowing one to find slow or large memory using statements. 419
Generates execution time and memory usage reports for GAMS statements. 468
Generates output on execution time and memory usage for GAMS statements 480
Option command that controls inclusion of statement execution time and memory use information. 551, 562
Profilefile
Name a file to recieve profile information 394
Profiletol
Command line parameter that specifies the minimum time a statement must use to appear in the profile generated output 394
Option command that specifies minimum execution time for inclusion of a statement in Profile output. 551, 563
Places lower limit on time use for statements in profile output and must be used carefully for memory searches 483
Places lower limit on time use for statements included in profile output. 470
Project
Difficulties with IDE project. 213
IDE storage file and file location definition. 177
Use of option command to project items. 555
Projected
Count of Levels set to bounds 93, 161
Levels reset when below tolproj 93
Ps
Changes length of output page in lines. 162
Command line parameter that tells the default number of lines per page. If less than 30 it will be reset to the default of 9999. 391
Customizable command parameter in Gmsprm file that specifies page size. If less than 30 it will be reset to 9999. 419
Ptol
Command line parameter that specifies the minimum time a statement must use to appear in the profile generated output 394
Put
Adding tabs 575, 602, 618, 701, 704
Command to assign current file and write to window. 618
Item formatting. 575
Rutherford's preprogrammed put files. 704
Using to pass data to other programs. 701
Put_utility
Add clickable link in process window 696
Add message to Log and LST file 696
Change put file name 696
Execute programs with strings as parameters 696
Execute with and without GAMS waiting for program to finish 696
Include file in put file 696
Reading multiple GDX files 696
Putclear
Command to remove all put headers and titles. 604
Putclose
Command to close file. 620
Puthd
Command to write to header block. 604
Putopen
Keyword in compile time $If to see if a put file is active. 655
Putpage
Command to assign current file and write to window with form feed. 621
Puttl
Command to write to title block. 603
Pw
Changes width of output page in columns. 162
Command line parameter that tells the number of columns on a page. 392
Customizable command parameter in Gmsprm file that specifies print width. 419

- Q -

QCP

© 2022 Prof. Bruce McCarl

QCP
Command line parameter that names QCP solver 395
Quadratically constrained programming model form. 100

- R -

R
Abbreviation for restart. 413
Command line parameter that gives the name of the restart file. 396
RandBinomial
Random numbers from binomial distribution 256
randLinear
Function for a random number from a linear distribution 256
Random number 256
Function to generate a random normal number. 256
Function to generate uniform random number. 256
Function to reset seed or retrieve seed for random number generator. 253
Generation from probability disributions 253, 256, 275, 564
Option command that specifies random number seed. 564
randTriangle
Function for a random number from triangular distribution 256
Ranging analysis
Cost and RHS ranging. 156
Rank
Ranking parameters in GDX files 304, 799
Ruterford and van der Eijk procedure to sort GAMS data. 304
Rdim
Rearranging placement of rows and columns when writing from Gdxxrw into spreadsheets. 744
Total dimension of item in rows in Gdxxrw data specification. 729
ReadyCollect
Function to collect grid and async jobs 874, 875
real1-5
Real communication cell 847
Rectangle
Moving rectangular blocks in middle of line with IDE. 195
Red line
Colored navigation line in IDE. 183
Reference
Command line parameter that gives the name of the file to receive extensive reference map information. 395
reform
Model attribute for reformularion 847
Refreader
Program for unraveling complex files in IDE. 202
rel_eq
Function that tests equality 260
rel_ge
Function that tests whether an iten is >= another 260
rel_gt
Function that tests whether an iten is >another 260
rel_le
Function that tests whether an iten is <= another 260
rel_lt
Function that tests whether an iten is < another 260
rel_ne 260
Ren
Put_utility command to change active put file 696
Repeat
Statement that executes multiple lines repetitively until a conditional is true. 317
Repeated static
Nature of calculations in Loop, For or While statements. 249
Replace
Keyword for Solveopt option causing replacement of solution information. 567
Solveopt that causes solution replacement. 97
Report writer
Cross scenario reports. 345
Report writing
Tutorial coverage 42
resGen
Model attribute - Model generation time 851
Reslim
Command line parameter specifies the maximum time in seconds that the computer can run during execution of a solver 395

© 2022 Prof. Bruce McCarl

Reslim
Expand maximum seconds job can execute. 440
Expand time with ETLIM 373, 440, 551, 563
Model attribute 847
Option command that specifies maximum seconds job can execute. 551, 563
Restart
Command line parameter that gives the name of the restart file. 396
GAMS command line parameter that restarts from a work file. 413
resUsd
Model attribute - Model solution time (only solver) 851
Rf
Command line parameter that gives the name of the file to receive extensive reference map information. 395
RHS ranging
Including output on RHS ranging. 156
RMINLP
Command line parameter that names the RMINLP solver. 396
Customizable command parameter in Gmsprm file that names RMINLP solver. 419
Declaration of model type specifying a relaxed mixed integer nonlinear program. 93
Keyword used to identify RMINLP solver name. 835
Option command that names RMINLP solver. 551, 564
Relaxed mixed integer nonlinear programming model form. 104
RMIP
Command line parameter that names the RMIP solver. 396
Customizable command parameter in Gmsprm file that names RMIP solver. 419
Declaration of model type specifying a relaxed mixed integer program. 93
Keyword used to identify RMIP solver name. 834
Option command that names RMIP solver. 551, 563
Relaxed mixed integer programming model form. 101
RMIQCP
Command line parameter that names RMIQCP solver 396
Mixed integer quadratically constrained programming model form. 105
Rng
Specifying a range in Gdxxrw. 727
rObj
Model attribute - Objective function value from the relaxed solve if the MIP solver did not finish 851
Round
Function to round numbers. 255
Rounding function. 297
rPower
Function that exponentiates an item to a power 256
Run button
Button to execute GAMS in IDE. 181
RunMacros
Specify how links in a spreadsheet should be updated during Gdxxrw operations. 758
Running a job
Tutorial coverage 14

- S -

S
Abbreviation for save. 413
Command line parameter that gives the name of save file. 397
Sameas
Comparing text defining set elements. 68
Function that tests if text for set elements match and false otherwise. 323
Save
ALtering names in save file 397, 405, 413
Command line parameter that gives the name of save file. 397
GAMS command line parameter that saves a binary format work file. 413
Save and restart
Save restart strategy to help isolate problem code in model debugging. 464
Savepoint
Command line parameter that causes a point GDX file to be saved with the current solution point. 397
Creating a GDX solution point file in GAMS 669
Customizable command parameter in Gmsprm file that causes a point GDX file to be saved with the current solution point. 419
Model Attribute 397, 419, 551, 564, 669, 819

© 2022 Prof. Bruce McCarl

Savepoint
Option command that causes a point GDX file to be saved with the current solution point. 551, 564
Saving a basis/current solution file 819
SBB
Mixed Integer Nonlinear solver 132, 833
Scalar
Command to define an item that is not dependent on sets 72
Scalar format
Using the CONVERT solver to transform a GAMS problem to a GAMS scalar format type of problem. 712
Scalar model
Definition and creation by CONVERT 785
Scalars
Command to define a data item that is not dependent on sets 72
Tutorial coverage 30
scaleOpt
Model attribute that tells Gams whether to use scaling factors 847
Scaling
Exercise of trying to reduce disparity of numbers in a model. 451
Improving scaling of nonlinear models 841
Scenario
Modifier to Solve statement 118
Scenario analysis
Addressing scenario analysis using the DEA solver. 116
Using GUSS 116, 118
SCENRED
A tool for scenario reduction in stochastic programming. 132
SCIP
Constrained Integer Programming solver 132
Scntr
Command line parameter that specifies the solver control file name 402
Scoped
Type of control variable defined only in parts of code. 629
Scrdir
Command line parameter that gives the name of the directory to be used by GAMS for temporary files generated during execution. 398
Screen
Directing put output to console screen. 579
Putting execution location to console or screen. 352
Scrext
Command line parameter that gives the name of the extension for the GAMS temporary files generated during execution 398
Script
Command line parameter that specifies the script mailbox file name 399
GAMS run from predefined script 210
Scriptexit
Command line parameter that specifies an exit script 398
Scriptfrst
Command line parameter that specifies the first line written to gamsnext 398
Scriptnext
Command line parameter that specifies the script mailbox file name 399
Scrnam
Command line parameter that gives the name stem to complete the names of intermediate work files 399
Sd
Command line parameter that gives the name of the directory to be used by GAMS for temporary files generated during execution. 398
Sdict
Command line parameter that specifies the solver dictionary file name 402
Sdir
Change where Sysinclude files come from. 498
Command line parameter that gives the directory where sysinclude files are kept. 406
Se
Command line parameter that gives the name of the extension for the GAMS temporary files generated during execution 398
Skipping blank rows or columns in Gdxxrw. 746
Seed
Command line parameter that specifies the random seed number 399
Option command that specifies random number seed. 551, 564
Self-documenting nature
Tutorial coverage 47
Semicont variable(s)
Command declaring variable as semi-continuous. 77
Semicont variables

© 2022 Prof. Bruce McCarl

Semicont variables
Variables that are zero or continuous above a threshold value. 826
Semiint variable(s)
Command declaring variable as semi-integer. 77
Semiint variables
Variables that are zero or are integer above a threshold value. 827
Sends specified text to the LOG file.
Using $log to send text to log file 165
Sensitivity analysis
Addressing sensitivity analysis using the DEA solver. 116
Cost and RHS ranging. 156
Set
Command that specifies a grouping of named elements. 49
Group of indices. 242
Inferring contents from a GDX file 49, 242, 525, 731, 738
Reading and writing sets in Gdxxrw with spreadsheets. 731
Reading sets from columns from spreadsheets in Gdxxrw. 738
Reading sets from rows from spreadsheets in Gdxxrw. 738
Singleton set - one element 49, 51, 242, 525, 731, 738
Set attributes
Set attribute giving position, length and value of set elements 64
Set element
Names of set elements 232
Set element text
Text which explains an item or set element 233
Set table
Using table command to define set elements 55
Set under control
Set inside set varying statement like sum or loop or a set on right hand side of statement indexed on left hand side over that set. 223
Set. 29, 30
Defining parameter values for all elements sof a set 74
Way of addressing an entire set when defining set elements 51
Setenv
Use in $IF to test for existence of environmental variables. 644
Sets
Command that specifies a grouping of named elements. 49
Loading from GDX files into GAMS. 741
Tips on defining 170
Tutorial coverage 28
Unloading into GDX files from GAMS. 742
Settype
Keyword in compile time $If to see if a named item is a set. 648
Sf
Command line parameter that specifies the first line written to gamsnext 398
Shademap
A tool for mapping GAMS results 711
Shared workbooks
Issues in Gdxxrw. 749
Shell
Put_utility command to invoke shell processor 696
Shellexecute
Executes external program chosen by operating system 808
Shuffle
Option to randomly rearrage a parameter 564
Sigmoid
Function to compute sigmoid. 256
Sign
Function to find sign of a number. 256
signPower
Signed power 256
Sin
Function to find sine. 256
Singleton set
Set with a sinlge element 51
Sinh
Function to find hyperbolic sine. 256
Sinst
Command line parameter that specifies the solver instruction file name 402
Skipempty
Skipping blank rows or columns in Gdxxrw. 746
Sl
Command line parameter that controls whether the GAMS program stays open during a solve. 400
Sleep(sec)
Function that causes GAMS to pause 263

© 2022 Prof. Bruce McCarl

slexp
Function giving linearly smoothed exponentiation 256
SLICE
Solving slice problems using DEA and CPLEX. 116
sllog10
Function for linearly smoothed log base 10 256
slrec
Function giving linearly smoothed reciprocal 256
Small to large
Modeling strategy to help in model debugging. 463
Smatr
Command line parameter that specifies the solver matrix file name 403
Smax
Function to find a maximum value over a set. 255
Smin
Function to find a minimum value over a set. 255
Sn
Command line parameter that gives the name stem to complete the names of intermediate work files 399
SNOPT
A LP, NLP, DNLP, RMIP, and RMINLP model type solver. 133
Solprint
Command line parameter that controls the printing of the solution report 399
Model attribute 847
Option command that suppresses solution printout in LST file. 551, 566
Solprint constants 155, 400, 551, 566
Suppresses solution in the output. 155
Solslack
Option and command line parameter that adds slack variable report to output. 156
Option command that includes slacks in solution output. 551, 567
Solution characteristics
Use of EXAMINER utility to examine solution adequacy. 117
Solution output
Differences in output when MCP models are solved 845
Solution Summary
Tutorial coverage 21
Solve
Command causing GAMS to invoke a solver. 93
Modification for GUSS 12, 93, 118
Tutorial coverage 12
Solvelink
Command line parameter that controls whether the GAMS program stays open during a solve. 400
Constants 401, 551, 565
Model attribute controling GAMS solver function 400, 565, 847
Option command that controls whether the GAMS program stays open during a solve. 565
Solveopt
Option command and model attribute controlling handing of solution information. 97, 847
Option command that controls way solution is placed into storage. 551, 567
Solver
GAMS use of solvers. 95
Making a solver default for all applicable model types 402, 567
Solver capabilities matrix
Solver ability capability. 108
Solver choice
Choosing the solver to use. 134
Solver choice in GAMS and IDE. 214
Solver documentation
Accessing solver documentation in the IDE. 200
Solver manuals 865
Accessing manuals 816
Places where solver options are fully defined. 816
Solver report
Tutorial coverage 21
Solver versions
Explanation of naming of multiple solvers. 109
Solvercntr
Command line parameter that specifies the solver control file name 402
Solverdict
Command line parameter that specifies the solver dictionary file name 402
Solverinst
Command line parameter that specifies the solver instruction file name 402
Solvermatr

© 2022 Prof. Bruce McCarl

Solvermatr
Command line parameter that specifies the solver matrix file name 403
Solvers
MCP and NLP solvers 846
Solversolu
Command line parameter that specifies the solver solution file name 403
Solverstat
Command line parameter that specifies the solver status file name 403
Solvestat
Constants 855
Model attribute - Solver termination condition 851
Solver solution status 855
SOPLEX
Another name for OSISOPLEX 133
Solver for LP problems 133
Sorteduels
Printing the sorted order of an unordered set. 67
Sorting
Methods to sort data in GAMS. 303
SOS1 variable(s)
Command declaring variable as one of a group of variables only one of which can be non zero. 77
SOS1 variables
Variables in groupings where only one variable in the group can be nonzero. 824
SOS2 variable(s)
Command declaring variable as one of a group of variables only two adjacent ones of which can be non zero. 77
SOS2 variables
Variables in groupings where only two variables in the group can be nonzero they must be adjacent. 825
Sp
Command line parameter that causes a point GDX file to be saved with the current solution point. 397
Speed
Finding and fixing speed problems 466
Speed implications of conditional placement. 317
Spell checking
Spell checking in GAMS IDE 210
SplitOption
Using parameters when executing external programs 549
Spreadsheet
Running GAMS from a spreadsheet 769
Spreadsheet graphics
Use of through Gdxxrw. 759
Spreadsheets
Interactively including results during a GAMS run. 761
Passing data to and from Excel spreadsheets. 721
sqexp
Function giving quadratically smoothed exponentiation 256
SQL
Interfacing GAMS with SQL based databases. 717
Sql2gms
Interfacing GAMS with SQL based databases. 717
sqlog10
Function giving quadratically smoothed logarithm base 10 256
Sqr
Function to square a number. 255
sqrec
Function giving quadratically smoothed reciprocal 256
Sqrt
Function to find square root of a number. 255
Squeeze
Writing of zero or default entries for attributes of variables and equations into spreadsheets with Gdxxrw. 753
Ssolu
Command line parameter that specifies thesolver solution file name 403
Sstat
Comamnd line parameter that specifies the solver status file name 403
Starting points
Values provided for the decision variables within the problem 838
Static
Nature of calculations in = statement. 249
Static calculations
Static calculations, data buildup and comparative analysis. 347
Stepsum

© 2022 Prof. Bruce McCarl

Stepsum
Command line parameter that controls creation of a step summary file 403
Stochastic programming
SCENRED tool for scenario reduction. 132
Strategic subsetting
Set use strategy to help in model debugging. 465
Strictsingleton
Optin controlling handling of multiple entries in singletons 404
Stringchk
Command line parameter that tells GAMS how to perform a string substitution check for %xxx% symbols. 404
Subset
Concept where one set is a subset of another. 324
Set made up of all or part of elements of another set. 53
Use of subsets to address portion of a set. 246
Subsets
Improving data input through subset use 170
Subsys
Command line parameter that gives configuration file name that contains solver defaults and other information. 405
Subsystems
Option command that causes GAMS to list all solvers and current default solvers in LST file. 551, 568
Sum
Function to add over a set. 256
SumInfes
Model attribute - Sum of infeasibilities 851
Sums
Tutorial coverage 25
Superbasic
Variable in NLP above and beyond number that would be basic 837
Suppress
Command line parameter that tells GAMS whether to suppress the compiler echo print. 405
Symbol
Command line parameter that gives name of the symbol table written in conjunction with reference files. 405
Symbol list
List of all named items known to the program in order and capitalization that they will appear in output. 306
Tutorial coverage 17
Symprefix
Prefix for items in save file 405
Syntax coloring
IDE color coding of GAMS syntax. 195
Sys10 559
Command line option 405, 551, 568
Option command that alters exponentiation. 405, 551, 568
Sysdir
Command line parameter that gives the name of the directory where the GAMS executable resides. 406
Sysincdir
Command line parameter that gives the directory where sysinclude files are kept. 406
Sysout
Adds additional solver reporting. 154
Command line parameter that controls the incorporation of additional solver generated output to the listing file 407
Model attribute 154, 569, 847
Option command that adds solver status file to LST file. 551, 569
System attributes
Use in put files. 589
System default settings
Way of customizing GAMS function. 141

- T -

Tab
Including tabs in put files 602
Tabin
Command line parameter that tells GAMS how to deal with tabs. 407
Table
Command to define a data item that is dependent on two or more sets 74
Use of table command to define set elements 55
Tables
Tutorial coverage 31
Tan(x)
Function giving tangent of x 256
Tanh

© 2022 Prof. Bruce McCarl

Tanh
Function to find hyperbolic tangent. 256
Termination messages
Influence of NLPs 845
Tf
Command line parameter that controls time format. 407
Tformat
Command line parameter that controls time format. 407
Customizable command parameter in Gmsprm file that identifies time format. 419
Threads
Command line parameter that controls the number of threads to be used by a solver 407
Controls use of multip CPU cores 407
Model attribute 407, 847
ThreadsAsync
Number of threads to reserve for Async computing 408
Time
Calendar, date and time functions 261
Timeclose
Function to recover the model closing time. 263
Timecomp
Function to recover the model compilation time. 263
Timeelapsed
Function to recover elapsed execution time. 263
Timeexec
Function to recover the model execution time. 263
Timer
Comamnd line parameter that specifies an instruction timer threshold 409
Timestart
Function to recover the model startup time. 263
Title
Put_utility command to change title of DOS window 696
To
Command in for statement indicating upper limit on scalar that is increased as it is varied. 339
Tolinfeas
Model attribute on infeasibility tolerance for an empty row 847
Tolinfrep
Model attribute giving infeasibility tolerance 847
tolProj
Model attribute giving tolerance on when a level gets projected to a bound and marginals are set to 0 when reading a solution 847
Number of levels set to lower or upper bound 93
Trace
Command line parameter that controls name and writing of a trace file. 409
Controls amount of amount of information in Gdxxrw log file. 757
Traceopt
Command line parameter that controls format of a trace file. 409
Trunc
Function to truncate a number. 256
Tryint
Model attribute to try a partial or near-integer solution 847
TryLinear
model attribute 847
Sees if MINLP, or MIQCP can be solved as MIP 90, 93
Sees if NLP, DNLP or QCP can be solved as LP 93
Sees if RMINLP, RMIQCP can be solved as RMILP 93
Tuple
Calculating a tuple 62, 246, 330, 741
Concept where a multi dimensional set is used to encode a conditional. 330
Defining using # and set names 62, 246, 330, 741
Defining with : matching operator 62, 246, 330, 741
Reading tuples from spreadsheets with Gdxxrw. 741
Set that is defined with respect to more that one other set and can be used in controlling sets. 246
Use in GUSS 118
Using tuples 62, 246, 330, 741

- U -

U1 to U5
Command line parameter that permits entry of text for up to 5 user defined options. 409
Using user defined options. 659
Uel
List of all set element names in model. 67

© 2022 Prof. Bruce McCarl

Uel
List of all set elements names known to the program in order and capitalization that they will appear in output. 307
Uncontrolled set
Set in a statement that is not subject to set varying statement like sum or loop or a set on right hand side of statement that is not indexed on left hand side. 224
Undf
Special value depicting an undefined number. 283
Undfout 256
Writing Undf into spreadsheets with Gdxxrw. 753
Uniform
Function to generate uniform random number. 256
UniformInt
Function to generate an integer uniform random number 256
Union
Method to define total set of elements across sets. 71
Unique element list
List of all set element names in model. 67
List of all set elements names known to the program in order and capitalization that they will appear in output. 307
Universal set
List of all set elements names known to the program in order and capitalization that they will appear in output. 63
Universal sets
Caution against using in input 169
Unknown symbol
Indication that GAMS is looking for a declared named item and spelling of this one doesn't match known items. 227
Unordered set 67
Set that has ordered elements and cannot be used with Ord, leads and lags. 67
Using sorteduels for printing the sorted order of an unordered set. 67
Unpack software
IDE assistance to unpack software. 216
Unravel
Unraveling complex files in with refreader. 202
Updlinks
Specify how links in a spreadsheet should be updated during Gdxxrw operations. 758
Upper and lower bounds
Adding bounds to improve nonlinear solver performance 840
Use by Others
Tutorial coverage 49
User
Using user defined options (user1-user5). 659
User defined functions
Adding a custom function 268
User Model Library
Defining your own library 862
User1 to User5
Command line parameter that permits entry of text for up to 5 user defined options. 409
User1 to User5
Using user defined options. 659
Using
Component of model statement indicating model type. 93

- V -

Values=
Controls set elements and explanatory text read from spreadsheet 731
Var
Reading and writing variables to/from spreadsheets with Gdxxrw. 746
Variable
Quantity that can be manipulated in the solution of a model. 77
Variable attribute
Solution, bound and scaling factors for a variable. 79
Variable attributes
Use in put files. 588
Variable listing
Algebra use effects on, tutorial coverage 36
Tutorial coverage 19
Variable solution report
Algebra use effects on, tutorial coverage 36
Tutorial coverage 23
Variable table
Assigning values for variable attributes 81
Variable(s)
Command declaring variable as one with no restriction. 77
Variables
Algebraic content, tutorial coverage 32

© 2022 Prof. Bruce McCarl

Variables
Tutorial coverage 7
Vartype
Keyword in compile time $If to see if a named item is a variable. 648
vcPower
Function that takes a number to a power 256
Viennadag
Using the CONVERT solver to transform a GAMS problem to a ViennaDag type of problem. 712
Viewclose
Scripting command 210
VISUAL BASIC
Programming language that can run GAMS 785
Running GAMS from macros in Excel 776
VISUAL C++
Programming language that can run GAMS 785

- W -

Warnings
Command line parameter that specifies the maximum number of warnings allowed 410
Keyword in compile time $If to see if a compilation has been warning free so far. 654
Wdir
Command line parameter that gives the name of the working directory. 410
Web sites
Sources of additional information 868
While
Ending early with Break or Continue 317, 336, 337
Executes block of statements as long as conditional is true. 337
Statement that executes multiple lines repetitively while a conditional is true. 317
Wiki 867
Interface Wiki 866
Support Wili 866
Work
Giving a solver more memory. 440
Option command that controls solver memory availability. 551
Workdir
Command line parameter that gives the name of the working directory. 410
Workfactor
Command line parameter that gives a multiplier for the GAMS solver memory estimate. 410
Model attribute 847
Workspace
Command line parameter that specifies the workspace in Mb for a solver 410
Model attribute for solver work space 847

- X -

XA
A solver of LP, MIP and RMIP model types. 133
XAPAR
A parallel processor version of XA. 133
Xldump
Exporting data to Excel spreadsheets with Rutherford's utilities. 725
Xlexport
Exporting data to Excel spreadsheets with Rutherford's utilities. 723
Ximport
Importing data from Excel spreadsheets with Rutherford's utilities. 722
Xls2gms
A program to generate an include file from contents of an Excel spreadsheet. 713
XLSDUMP
Dump all data in a spreadsheet to GDX 716
Xlstalk
Utility that communicates with Excel 808
XML 712
Interfacing COINFML style XML with GAMS. 712
Interfacing GAMS with XML. 712
Xor
Relational operator that links sub-logical conditions being true when only one sub condition is true. 327
XPRESS
A LP, MIP, and RMIP problem solver. 133
Free bare bones version 133
Xs
Command line parameter that specifies the name of a save file written in ASCII format so it is platform independent. Note restart automatically will read this file type 411
Restart files transferable between operating systems. 414
Writing compressed restart files 411, 414

© 2022 Prof. Bruce McCarl

Xsave
Command line parameter that specifies the name of a save file written in ASCII format so it is platform independent. Note restart automatically will read this file type. 411
Restart files transferable between operating systems. 414
Writing compressed restart files 411, 414

- Y -

Yes
Defines presence of a set element in a calculation. 284
Special value that indicates set element membership, can be used in assignment statement. 56

- Z -

Zeroout
Writing zeros into spreadsheets with Gdxxrw. 753
Zerores 411
Command line parameter that specifies the threshold value for internal rounding to zero 411
Zeroresrep
Command line parameter that controls warnings when Zerores is used 411
Zeros
Skipping of zeros in display statements. 293
ZIP files
Utilities to ZIP and UNZIP 809

© 2022 Prof. Bruce McCarl