GAMS -The Solver Manuals

- **Using Solver Specific Options**
- **BDMLP**
- **CONOPT**
- **CPLEX**
- **DECIS**
- **DICOPT**
- **GAMSBAS**
- **GAMSCHK**
- **MILES**
- **MINOS**
- **MPSWRITE**
- **OSL**
- **OSL Stochastic Extensions**
- **PATH**
- **SBB**
- **SNOPT**
- **XA**
- **XPRESS**

March 2001

GAMS Development Corporation
1217 Potomac Street, N.W.
Washington, DC 20007, USA
Tel: +1-202-342-0180
Fax: +1-202-342-0181
E-mail:sales@gams.com
Http://www.gams.com/

GAMS Software GmbH
Eupener Str. 135-137
50933 Cologne, Germany
Tel.: +49-221-949-9170
Fax: +49-221-949-9171
E-mail:info@gams.de
Http://www.gams.de/

USING SOLVER SPECIFIC OPTIONS

INTRODUCTION

In addition to the general options available in the GAMS language, most solvers allow the user to affect certain alg11rithmic details and thereby affect their effectiveness. The default values for all the solver specific options are usually appropriate for most problems. However, in some cases, it is possible to improve the performance of a solver by specifying non-standard values for some of the options.

Please read the rele1vant section of the solver manual very carefully before using any of the solver options. Only advanced users who understand the implications of changing algorithmic parameters should attempt to use these options.

THE SOLVER OPTION FILE

If you want to have a solver use an option file, it is necessary to inform the solver in the GAMS model. Setting the `optfile` model suffix to a positive value will perform that task. For example,

```
model mymodel /all/ ;
mymodel.optfile = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The name of the option file that is specific to each solver. being used after this assignment is *solvername.XXX*, where *'solvername'* is the name of the solver that is specified, and the suffix *XXX* depends on the value to which the model suffix `optfile` has been set. If its value is 1, the suffix is *opt*. For example, the option file for CONOPT is called *conopt.opt*; for DICOPT, it is *dicopt.opt*.

If you do not set the `.optfile` suffix to 1, no option file will be used even if it exists.

To allow different option file names for the same solver, the `.optfile` model suffix can take other values as well. Formally, the rule is that a file named *solvername.opt* will be read if it is set to 1, and *solvername.opX*, *solvername.oXX* or *solvername.XXX*, where *X*'s are the characters representing its value. No option file will be read if it is set to 0. This scheme implies an upper bound on n of 999. For example,

`optfile` model suffix value	Name of option file
0	No option file used
1	*solvername*.opt
2	*solvername*.op2
3	*solvername*.op3
10	*solvername*.o10
91	*solvername*.o91
100	*solvername*.100
999	*solvername*.999

For example, setting `mymodel.optfile` to 23 will result in the option file *conopt.o23* being used for CONOPT, and *dicopt.o23* being used for DICOPT.

FORMAT OF THE SOLVER OPTION FILE

The format of the options file is not completely standard and changes marginally from solver to solver. This section illustrates some of the common features of the option file format. Please check the specific section of the solver manual before using the option file.

Each line of the option file falls into one of two categories

- a comment line
- an option specification line

A comment 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. Each option specification line can contain only one option. The format for specifying options is as follows,

```
keyword(s) [modifier]    [value]
```

The keyword may consist of one or more words and is not case sensitive. Numerical values of the keywords may be either an integer or a real constant. Real numbers may be expressed in `F`, `E`, or `D` formats. Note that not all options require modifiers or values.

Any errors in the spelling of the key word(s) or modifiers will lead in that option not being understood and therefore not being used by the solver.

Any errors in the values of the options will result in unpredictable behavior. The options is either ignored or set to a default or limiting value. A very careful check of the option values are needed before use.

Consider the following CPLEX options file,

```
* CPLEX options file
barrier
crossover 2
```

The first line begins with an asterisk and therefore contains comments. The first option specifies the use of the barrier algorithm to solver the linear programming problem, while the second option specifies that the crossover option 2 is to be used. Details of these options can be found in the CPLEX section of this manual.

Consider the following MINOS options file,

```
*MINOS options file
scale option 2
completion partial
```

The first option sets the scale option to a value of 2. In this case, the key word `'scale option'` consists of two words. In the second line, the `completion` option is set to `partial`. Details of these options can be found in the MINOS section of this manual.

GAMS/BDMLP User Notes

GAMS/BDMLP is a free LP and MIP solver that comes with any GAMS system. It is intended for small to medium sized models. GAMS/BDMLP was originally developed at the World Bank by T. Brooke, A. Drud, and A. Meeraus and is now maintained by GAMS Development. The MIP part was added by M. Bussieck and A. Drud. GAMS/BDMLP is running on all platform for which GAMS is available.

GAMS/BDMLP can solve reasonably sized LP models, as long as the models are not very degenerate and are well scaled. The Branch-and-Bound algorithm for solving MIP is not in the same league as other commercial MIP codes that are hooked up to GAMS. Nevertheless, the MIP part of GAMS/BDMLP provides free access to a MIP solver that supports all types of discrete variables supported by GAMS: Binary, Integer, Semicont, Semiint, Sos1, Sos2.

How to run a Model with BDMLP

GAMS/BDMLP can solve models of the following types: LP, RMIP, and MIP. If you did not specify BDMLP as the default LP, RMIP, or MIP solver, use the following statement in your GAMS model before the `solve` statement:

```
option lp=bdmlp; { or RMIP or MIP}
```

GAMS/BDMLP Options

Options can be specified through the option statement (iterlim, reslim, optca, and optcr) and through a model suffix. The option statement sets the GAMS option globally

```
option iterlim = 100 ;
```

where the model suffix sets the GAMS option for individual models:

```
mymodel.iterlim = 10;
```

The following GAMS options are used by GAMS/BDMLP:

Option	Description
iterlim	Sets the iteration limit. BDMLP will terminate and pass on the current solution to GAMS.
reslim	Sets the time limit in seconds. BDMLP will terminate and pass on the current solution to GAMS.
nodlim	Sets the branch and bound node limit. BDMLP will terminate and pass on the current solution to GAMS.
bratio	Determines whether or not to use an advanced basis.
sysout	Will echo the BDMLP messages to the GAMS listing file.
optca	Absolute optimality criterion. The optca option asks BDMLP to stop 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.
optcr	Relative optimality criterion. The optcr option asks BDMLP to stop when `(|BP-BF|)/(|BP|)< optcr`.
prioropt	Instructs BDMLP to use priority branching information passed by GAMS through the *variable*.prior parameters.
cheat	Cheat value: Each new integer solution must be at least <*cheat*> better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the optca option).
cutoff	Cutoff value: When the branch and bound search starts, the parts of the tree with an objective worse than <*cutoff*> are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.
tryint	Causes BDMLP to make use of current variable values. If a variable value is within <*tryint*> of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound.

GAMS/BDMLP does not support an option file and it has no additional user-settable options.

GAMS/CONOPT

ARKI Consulting and Development
Bagsvaerdvej 246A, DK-2880 Bagsvaerd, Denmark
Tel: +45 44 49 03 23, Fax: +45 44 49 03 33,Email: info@arki.dk

Table of Contents:

1. INTRODUCTION

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there are three standard NLP algorithms available, CONOPT, MINOS and SNOPT, and CONOPT is available in two versions, the old CONOPT and the new CONOPT2. All algorithms attempt to find a local optimum. The algorithms in CONOPT, MINOS, and SNOPT are all based on fairly different mathematical algorithms, and they behave differently on most models. This means that while CONOPT is superior for some models, MINOS or SNOPT will be superior for others. To offer modelers with a large portfolio of NLP models the best of all worlds, GAMS offers various NLP package deals consisting of two or three NLP solvers for a reduced price if purchased together.

Even CONOPT and CONOPT2 behaves differently; the new CONOPT2 is best for most models, but there are a small number of models that are best solved with the older CONOPT and the old version is therefore still distributed together with CONOPT2 under the same license. However, you should notice that the old CONOPT is no longer being developed, so if you encounter problems with CONOPT, please try to use CONOPT2 instead.

It is almost impossible to predict how difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot select the best algorithm for you automatically. One of the nonlinear programming algorithms must be selected as the default when GAMS is installed. If you want to choose a different algorithm or if you want to switch between algorithms you may add the statement `"OPTION NLP = CONOPT2;"`, `"OPTION NLP = CONOPT;"`, `"OPTION NLP = MINOS;"` or `"OPTION NLP = SNOPT;"` in your GAMS source file before the `SOLVE` statement, or you may rerun the installation program. The only reliable way to find which solver to use for a particular class of models is so far to experiment. However, there are a few rules of thumb:

GAMS/CONOPT2 is well suited for models with very nonlinear constraints. If you experience that MINOS has problems maintaining feasibility during the optimization you should try CONOPT2. On the other hand, if you have a model with few nonlinearities outside the objective function then either MINOS or SNOPT are probably the best solver.

GAMS/CONOPT2 has a fast method for finding a first feasible solution that is particularly

well suited for models with few degrees of freedom. If you have a model with roughly the same number of constraints as variable you should try CONOPT2. CONOPT2 can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System. If the number of variables is much larger than the number of constraints MINOS or SNOPT may be better.

GAMS/CONOPT2 has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one then CONOPT2 will take advantage of this property. Similarly, intermediate variables only used to define objective terms are eliminated from the model and the constraints are moved into the objective function.

GAMS/CONOPT2 has many built-in tests and messages, for example for poor scaling, and many models that can and should be improved by the modeler are rejected with a constructive message. CONOPT2 is therefore also a helpful debugging tool during model development. The best solver for the final, debugged model may or may not be CONOPT2.

GAMS/CONOPT2 has been designed for large and sparse models. This means that both the number of variables and equations can be large. Indeed, NLP models with over 20000 equations and variables have been solved successfully, and CNS models with over 500000 equations and variables have also been solved. The components used to build CONOPT2 have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT2 can also be used for denser models, but the performance will suffer significantly.

GAMS/CONOPT2 is designed for models with smooth functions, but it can also be applied to models that do not have differentiable functions, in GAMS called DNLP models. However, there are no guaranties whatsoever for this class of models and you will often get termination messages like "Convergence too slow" or "No change in objective although the reduced gradient is greater than the tolerance" that indicate unsuccessful termination. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form.

Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT2 has considerable build-in logic that selects a solution approach that seems to be best suited for the type of model at hand, and the approach is adjusted dynamically as information about the behavior of the model is collected and updated. The description of the CONOPT2 algorithm has therefore been moved to Appendix A and most modelers can skip it. However, if you are solving very large or complex models or if you are experiencing solution difficulties you may benefit from using non-standard tolerances or options, in which case you will need some understanding of what CONOPT2 is doing to your model. Some guidelines for selecting options can be found at the end of Appendix A and a list of all options and tolerances is shown in Appendix B.

The main text of this User’s Guide will give a short overview over the iteration output you will see on the screen (section 2), and explain the termination messages (section 3). We will then discuss function evaluation errors (section 4), the use of options (section 5), and

give a CONOPT2 perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section 6). Finally, we will discuss the difference between NLP and DNLP models (section 7). The text is mainly concerned with the new CONOPT2 but most of it will also cover the older CONOPT and we will use the generic name CONOPT when refering to the solver. Some features are only available in the new CONOPT2 in which case we will mention it explicitly. Messages in the old CONOPT may have a format that is slightly different from the one shown here.

2. ITERATION OUTPUT

On most machines you will by default get a logline on your screen or terminal at regular intervals. The iteration log may look something like this:

```
 Iter Phase Ninf   Infeasibility   RGmax   NSB  Step SlpIt MX OK
    0   0        8.3247946065E+02 (Input point)
                           Pre-triangular equations:       0
                           Post-triangular equations:      5
    1   0        9.1593000000E+01 (After pre-processing)
    2   0        9.1593000000E+01 (After scaling)
   10   0    14  9.1593000000E+01             0.0E+00      F  F
   21   1    17  4.8613463018E+01 1.3E+01  28 9.9E-03      T  T
   26   1    15  4.0564945133E+01 4.0E+00  22 1.4E-01   7  T  T
   31   1    12  3.0991342925E+01 3.1E+00  24 1.3E-01   5  F  F
   36   1    11  2.6438208207E+01 2.0E+00  20 5.0E-01   3  F  F
   41   1     8  1.9451475242E+01 1.6E+00  18 1.0E+00   1  F  T
   46   1     8  1.6816400505E+01 1.3E+00  16 3.2E-01   1  F  F

 Iter Phase Ninf  Infeasibility   RGmax  NSB   Step SlpIt MX OK
   51   1     7  1.2558236729E+01 1.0E+00  15 3.6E-01   4  F  F
   56   1     5  8.7085621669E+00 1.1E+00  13 6.0E-02   6  F  F
   61   1     3  6.3837868042E+00 1.0E+00  11 2.2E-01   1  F  F
   66   1     2  4.6815820465E+00 1.0E+00   8 2.2E-02   1  F  F
   71   1     2  3.0484549833E+00 1.0E+00   8 1.6E-01      F  F
   81   1     1  2.0098693785E-01 1.0E+00   7 1.6E-01      F  F

** Feasible solution. Value of objective =    1571.04999262

 Iter Phase Ninf   Objective      RGmax   NSB  Step SlpIt MX OK
   91   3        1.4598200364E+03 8.7E+01   6 1.2E-05      F  F
  101   3        1.3173744754E+03 7.1E+01   5 1.2E-03      F  F
  111   4        1.0406602405E+03 1.2E+02   6 3.0E-03      F  F
  121   4        9.3135586320E+02 1.0E+02   5 3.1E+00      F  F
  131   4        9.2501180675E+02 1.5E+00   4 2.1E+00      F  T
  141   4        9.2496527723E+02 2.7E-04   4 1.0E+00      F  T
  144   4        9.2496527723E+02 1.4E-08   4

** Optimal solution. Reduced gradient less than tolerance.
```

The first few iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represent the initial point after CONOPT's pre-processing, and iteration 2 represents the same point after scaling (even if scaling is turned off, which is the default).

The remaining iterations are characterized by the "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 are minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4, is minimized or maximized. Phase 0 iterations are cheap Newton-like iterations. During Phase 1 and 3 the model behaves almost linearly and special linear iterations that take advantage of the linearity are performed, sometimes augmented with some inner "Sequential Linear Programming" iterations, indicated by a number in the SlpIt column. During Phase 2 and 4 the model behaves more nonlinearly and most aspects of the iterations are therefore changed: the line search is more elaborate, and CONOPT estimates second order information to improve the convergence.

The column NSB for Number of SuperBasics defines the degree of freedom or the dimension of the current search space, and Rgmax measures the largest gradient of the non-optimal variables. Rgmax should eventually converge towards zero. The last two columns labeled MX and OK gives information about the line search: MX = T means that the line search was terminated by a variable reaching a bound, and MX = F means that the optimal step length was determined by nonlinearities. OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated because it was not possible to find a feasible solution for large step lengths.

---

3. GAMS/CONOPT TERMINATION MESSAGES

GAMS/CONOPT may terminate in a number of ways. This section will show most of the termination messages and explain their meaning. It will also show the Model Status returned to GAMS in `<model>.Modelstat`, where `<model>` represents the name of the GAMS model. The Solver Status returned in `<model>.Solvestat` will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation. The first 4 messages are used for optimal solutions and CONOPT will return Modelstat = 2 (Locally Optimal), except as noted below:

```
** Optimal solution. There are no superbasic variables.
```

The solution is a locally optimal corner solution. The solution is determined by constraints only, and it is usually very accurate. In some cases CONOPT can determine that the solution is globally optimal and it will return Modelstat = 1 (Optimal).

```
** Optimal solution. Reduced gradient less than tolerance.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the tolerance `rtredg` with default value around 1.d-7. The value of the objective function is very accurate while the values of the variables are less accurate due to a flat objective function in the interior of the feasible area.

```
** Optimal solution. The error on the optimal objective function
```

```
value estimated from the reduced gradient and the estimated
Hessian is less than the minimal tolerance on the objective.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the tolerance `rtredg`. However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. The objective must be large or the reduced objective must have large second derivatives so it is advisable to scale the model. See the sections on "Scaling" and "Using the Scale Option in GAMS" for details on how to scale a model.

```
** Optimal solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.
```

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance `rtredg`, but less than `rtredg` multiplied by the largest Jacobian element divided by 100. The model must have large derivatives so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and Modelstat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that a Sum of Infeasibility objective function is locally minimal, but positive. If the model is convex then it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on "Initial Values" for hints on what to do.

```
** Feasible solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.

** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all. However, the optimality criteria have not been satisfied. These messages are accompanied by Modelstat = 7 (Intermediate Nonoptimal) and Solvestat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section 7. The problem can also be caused by a poorly scaled model. See section 6.5 for hints on model scaling. Finally, it can be cause by stalling as described in section A12.4 in Appendix A. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". Modelstat is in this case 6 (Intermediate Infeasible) and Solvestat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well defined local minimum.

```
<var>: The variable has reached infinity
```

```
** Unbounded solution. A variable has reached 'infinity'.
   Largest legal value (Rtmaxv) is xx.xx
```

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value and it returns with Modelstat = 3 (Unbounded). Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable <var> mentioned in the first line of the message. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, `rtmaxv`, as mentioned in the section on options. However, you will pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from "infinity", CONOPT will actually return a feasible solution with large values for the variables.

The message above exist in a version where "Unbounded" is replaced by "Infeasible" and Modelstat is 5 (Locally Infeasible). You may also see a message like

```
   <var>: Free variable becomes too large

** Infeasible solution. A free variable exceeds the allowable
   range.  Current value is 4.20E+07 and current upper bound
   (Rtmaxv) is 3.16E+07
```

These two messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable <var> mentioned in the first line of the message. If the model seems correct you should scale it.

```
** The time limit has been reached.
```

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by `"OPTION RESLIM = xx;"` or `"<model>.RESLIM = xx;"` statements, has been reached. CONOPT will return with Solvestat = 3 (Resource Interrupt) and Modelstat either 6 (Locally Infeasible) or 7 (Locally Nonoptimal).

```
** The iteration limit has been reached.
```

The iteration limit defined in GAMS, either by default (usually 1000 iterations) or by `"OPTION ITERLIM = xx;"` or `"<model>.ITERLIM = xx;"` statements, has been reached. CONOPT will return with Solvestat = 2 (Iteration Interrupt) and Modelstat either 6 (Locally Infeasible) or 7 (Locally Nonoptimal).

```
** Domain errors in nonlinear functions.
   Check bounds on variables.
```

The number of function evaluation errors has reached the limit defined in GAMS by `"OPTION DOMLIM = xx;"` or `"<model>.DOMLIM = xx;"` statements or the default limit of 0 function evaluation errors. CONOPT will return with Solvestat = 5

(Evaluation Error Limit) and Modelstat either 6 (Locally Infeasible) or 7 (Locally Nonoptimal). See section 4 for more details on "Function Evaluation Errors".

```
** An initial derivative is too large (larger than Rtmaxj= xx.xx)
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Initial Jacobian element too large = xx.xx
```

and

```
** A derivative is too large (larger than Rtmaxj= xx.xx).
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Jacobian element too large = xx.xx
```

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will show you where to look. A large derivative means that the function changes very rapidly with changes in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop and you are advised to adjust the model if at all possible.

If the offending derivative is associated with a LOG(X) or 1/X term you may try to increase the lower bound on X. If the offending derivative is associated with an EXP(X) term you must decrease the upper bound on X. You may also try to scale the model, either manually or using the variable.SCALE and/or equation.SCALE option in GAMS as described in section 6.5. There is also in this case a lazy solution: increase the limit on Jacobian elements, `rtmaxj`; however, you will pay through reduced reliability or longer solution times.

In addition to the messages shown above you may see messages like

```
** An equation in the pre-triangular part of the model cannot be
   solved because the critical variable is at a bound.

** An equation in the pre-triangular part of the model cannot be
   solved because of too small pivot.
```

or

```
** An equation is inconsistent with other equations in the
   pre-triangular part of the model.
```

These messages containing the word "Pre-triangular" are all related to infeasibilities identified by CONOPT’s pre-processing stage and they are explained in detail in section A4 in Appendix A.

Usually, CONOPT will be able to estimate the amount of memory needed for the model based on statistics provided by GAMS. However, in some cases with unusual models, e.g.

very dense models or very large models, the estimate will be too small and you must request more memory yourself using a statement like "<model>.WORKSPACE = xx;" in GAMS. The message you will see is similar to the following:

```
** FATAL ERROR **  Insufficient memory to continue the
                   optimization.

                   You must request more memory.
                   Current   CONOPT space =  0.29 Mbytes
                   Estimated CONOPT space =  0.64 Mbytes
                   Minimum   CONOPT space =  0.33 Mbytes


CONOPT time Total                          0.109 seconds
  of which: Function evaluations           0.000 =  0.0%
            Derivative evaluations         0.000 =  0.0%

Work length =   45875 double words  =  0.35 Mbytes
   Estimate =   45875 double words  =  0.35 Mbytes
   Max used =   45627 double words  =  0.35 Mbytes
```

The text after "Insufficient memory to" may be different; it says something about where CONOPT ran out of memory. If the memory problem appears during model setup the message will be accompanied by Solvestat = 9 (Error Setup Failure) and Modelstat = 13 (Error No Solution) and CONOPT will not return any values. If the memory problem appears later during the optimization Solvestat will be 10 (Error Internal Solver Failure) and Modelstat will be either 6 (Intermediate Infeasible) or 7 (Intermediate Nonoptimal) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

The first set of statistics in the message text shows you how much memory is available for CONOPT, and the last set shows how much is available for GAMS and CONOPT combined (GAMS needs space to store the nonlinear functions). The GAMS WORKSPACE option corresponds to the combined memory, measured in Mbytes, so you should concentrate on the last numbers.

---

4. FUNCTION EVALUATION ERRORS

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. LOG is not defined for negative arguments, EXP overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable bounds on your variables. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds.

In some cases bounds are not sufficient, e.g. in the expression LOG(SUM(I, X(I))): in some models each individual X should be allowed to become zero, but the SUM should not. In this case you should introduce an intermediate variable and an extra equation, e.g. XSUMDEF .. XSUM =E= SUM(I,X(I)); add a lower bound on XSUM; and use XSUM as

the argument to the LOG function. See section 6.3 on "Simple Expressions" for additional comments on this topic.

Whenever a nonlinear function is called outside its domain of definition, GAMS' function evaluator will intercept the function evaluation error and prevent that the system crashes. GAMS will replace the undefined result by some appropriate real number, and it will make sure the error is reported to the modeler as part of the standard solution output in the GAMS listing file. GAMS will also report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Finally, CONOPT will be instructed to stop after DOMLIM errors.

During Phase 0, 1, and 3 CONOPT will often use large steps as the initial step in a line search and functions will very likely be called with some of the variables at their lower or upper bound. You are therefore likely to get a division-by-zero error if your model contains a division by X and X has a lower bound of zero. And you are likely to get an exponentiation overflow error if your model contains EXP(X) and X has no upper bound. However, CONOPT will usually not get trapped in a point outside the domain of definition for the model. When GAMS' function evaluator reports that a point is "bad", CONOPT will decrease the step length, and it will for most models be able to recover and continue to an optimal solution. It is therefore safe to use a large value for DOMLIM instead of GAMS default value of 0.

CONOPT may get stuck in some cases, for example because there is no previous point to backtrack to, because "bad" points are very close to "reasonable" feasible points, or because the derivatives are not defined in a feasible point. The more common messages are:

```
** Fatal Error **  Function error in initial point in Phase 0
                    procedure.

** Fatal Error **  Function error after small step in Phase 0
                    procedure.

** Fatal Error **  Function error very close to a feasible point.

** Fatal Error **  Function error while reducing tolerances.

** Fatal Error **  Function error in Pre-triangular equations.

** Fatal Error **  Function error after solving Pre-triangular
                    equations.

** Fatal Error **  Function error in Post-triangular equation.
```

In the first four cases you must either add better bounds or define better initial values. If the problem is related to a pre- or post-triangular equation as shown by the last three messages then you can turn part of the pre-processing off as described in section A4 in Appendix A. However, this may make the model harder to solve, so it is usually better to add bounds and/or initial values.

5. THE CONOPT OPTIONS FILE

CONOPT has been designed to be self-tuning. Most tolerances are dynamic. As an example: The feasibility of a constraint is always judged relative to the dual variable on the constraint and relative to the expected change in objective in the coming iteration. If the dual variable is large then the constraint must be satisfied with a small tolerance, and if the dual variable is small then the tolerance is larger. When the expected change in objective in the first iterations is large then the feasibility tolerances are also large. And when we approach the optimum and the expected change in objective becomes smaller then the feasibility tolerances become smaller.

Because of the self-tuning nature of CONOPT you should in most cases be well off with default tolerances. If you do need to change some tolerances, possibly following the advice in Appendix A, it can be done in the CONOPT Options file. The name of the CONOPT Options file is on most systems `"conopt2.opt"` for the new CONOPT2 and `"conopt.opt"` for the old CONOPT. You must tell the solver that you want to use an options file with the statement <model>.OPTFILE = 1;, in your GAMS source file before the SOLVE statement.

The format of the CONOPT Options file is different from the format of options file used by MINOS and ZOOM. It consists in its simplest form of a number of lines like these:

```
rtmaxv :=  1.e8;
lfnsup := 500;
```

An optional `"set"` verb can be added in front of the assignment statements, and the separators `:`, `=`, and `;` are silently ignored, so the first line could also be written as `"set rtmaxv 1.e8"` or simply `"rtmaxv 1.e8"`. Upper case letters are converted to lower case so the second line could also be written as `"LFNSUP := 500;"`. The assignment or `set` statement is used to assign a new value to internal CONOPT variables, so-called CR-Cells. The optional `set` verb, the name of the CR-Cell, and the value must be separated by blanks, tabs, commas, colons, and/or equal signs. The value must be written using legal Fortran format with a maximum of 10 characters, i.e. a real number may contain an optional E and D exponent, but a number may not contain blanks. The value must have the same type as the CR-Cell, i.e. real CR-Cells must be assigned real values, integer CR-Cells must be assigned integer values, and logical CR-Cells must be assigned logical values. Values can also be the name of a CR-Cell of the same type.

The `crprint` statement with syntax `"crprint crcell1 crcell2 ..;"` is used to print the current value of one or more CR-Cells. The output of CRPRINT can only be seen in the Solver Status File that GAMS echoes back when you use "`OPTION SYSOUT = ON`".

The `step` statement with syntax `"step crcell value;"` can be used to increment a CR-Cell by a certain amount. "Value" can be a number or a CR-Cell of the same type. For example, the statement `"step rtredg rtredg;"` will double the value of `rtredg`.

The CONOPT Options file can be used for many other purposes not needed with GAMS, e.g. for certain iterative constructs and for SAVE and RESTART. These constructs are mainly used with the Subroutine Library version of CONOPT and for debugging and they are not covered here.

6. HINTS ON GOOD MODEL FORMULATION

This section will contain some comments on how to formulate a nonlinear model so it becomes easier to solve with CONOPT. Most of the recommendations will be useful for any nonlinear solver, but not all. We will try to mention when a recommendation is CONOPT specific.

6.1 Initial Values

Good initial values are important for many reasons. Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution. Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort. 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 the initial point in which the initial derivatives are computed is therefore again important. Finally, non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

The initial values used by CONOPT are all coming from GAMS. The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero. Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. 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. CONOPT will warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.

If a variable has a small positive lower bound, for example because it appears as an argument to the LOG function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.

You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS. An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS' scaling option. A better possibility is to select reasonable values for some variables that from the context are known to be

important, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

```
PMDEF(IT) .. PM(IT) =E= PWM(IT)*ER*(1 + TM(IT)) ;
```

where PM, PWM, and ER are variables and TM is a parameter. The following assignment statements use the equation to derive consistent initial values for PM from sensible initial values for PWM and ER:

```
ER.L = 1; PWM(IT) = 1;
PM.L(IT) = PWM.L(IT)*ER.L*(1 + TM(IT)) ;
```

With these assignments equation PMDEF will be feasible in the initial point, and since CONOPT uses a feasible path method it will remain feasible throughout the optimization (unless the pre-processor destroys it, see section A4 in Appendix A).

If CONOPT has difficulties finding a feasible solution for your model you should try to use this technique to create an initial point in which as many equations as possible are satisfied. If you use CONOPT2 you may also try the optional Crash procedure described in section A4.3 in Appendix A by adding the line `"lstcrs=t"` to the CONOPT2 options file. The crash procedure tries to identify equations with a mixture of un-initialized variables and variables with initial values, and it solves the equations with respect to the un-initialized variables; the effect is similar to the manual procedure shown above.

6.2 Bounds

Bounds have two purposes in nonlinear models. Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems. Algorithmic bounds require a closer look at the functional form of the model. The content of a LOG should be greater than say 1.e-3, the content of an EXP should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both LOG(X) and 1/X are extremely nonlinear for small arguments. The first and second derivatives of LOG(X) at X=1.e-3 are 1.e+3 and -1.e6, respectively, and the first and second derivatives of 1/X at X=1.e-2 are -1.e+4 and 2.e+6, respectively.

If the content of a LOG or EXP function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed in the next section.

Note that bounds in some cases can slow the solution process down. Too many bounds may for example introduce degeneracy. If you have constraints of the following type

```
VUB(I) .. X(I) =L= Y;
```

or

```
YSUM   .. Y =E= SUM( I, X(I) );
```

and X is a POSITIVE VARIABLE then you should in general not declare Y a POSITIVE VARIABLE or add a lower bound of zero on Y. If Y appears in a nonlinear function you may need a strictly positive bound. Otherwise, you should declare Y a free variable; CONOPT will then make Y basic in the initial point and Y will remain basic throughout the optimization. New logic in CONOPT2 tries to remove this problem by detecting when a harmful bound is redundant so it can be removed, but it is not yet a fool proof procedure.

Section A4 in Appendix A gives another example of bounds that can be counter productive.

6.3 Simple Expressions

The following model component

```
PARAMETER MU(I);
VARIABLE  X(I), S(I), OBJ;
EQUATION  OBJDEF;
OBJDEF .. OBJ =E= EXP( SUM( I, SQR( X(I) - MU(I) ) / S(I) ) );
```

can be re-written in the slightly longer but simpler form

```
PARAMETER MU(I);
VARIABLE  X(I), S(I), OBJ, INTERM;
EQUATION  INTDEF, OBJDEF;
INTDEF .. INTERM =E= SUM( I, SQR( X(I) - MU(I) ) / S(I) );
OBJDEF .. OBJ    =E= EXP( INTERM );
```

The first formulation has very complex derivatives because EXP is taken of a long expression. The second formulation has much simpler derivatives; EXP is taken of a single variable, and the variables in INTDEF appear in a sum of simple independent terms.

In general, try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables. Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable. The model will become larger, but the increased size is taken care of by CONOPT's sparse matrix routines, and it is compensated by the reduced complexity.

The reduction in complexity can be significant if an intermediate expression is linear. The following model fragment:

```
VARIABLE X(I), Y;
EQUATION YDEF;
YDEF ..  Y =E= 1 / SUM(I, X(I) );
```

should be written as

```
VARIABLE X(I), XSUM, Y;
EQUATION XSUMDEF, YDEF;
XSUMDEF .. XSUM =E= SUM(I, X(I) );
YDEF    .. Y =E= 1 / XSUM;
XSUM.LO = 1.E-2;
```

for two reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. And second, because the lower bound on the intermediate result will bound the search away from the singularity at XSUM = 0.

The last example shows an added potential saving by expanding functions of linear expressions. A constraint depends in a nonlinear fashion on the accumulated investments, INV, like

```
CON(I) .. f( SUM( J$(ORD(J) LE ORD(I)), INV(J) ) ) =L= B(I);
```

A new intermediate variable, CAP(I), that is equal to the content of the SUM can be defined recursively with the constraints

```
CDEF(I) .. CAP(I) =E= INV(I) + CAP(I-1);
```

and the original constraints become

```
CON(I) .. f( CAP(I) ) =L= B(I);
```

The reformulated model has N additional variables and N additional linear constraints. In return, the original N complex nonlinear constraints have been changed into N simpler nonlinear constraints. And the number of Jacobian elements, that has a direct influence on much of the computational work both in GAMS and in CONOPT, has been reduced from N*(N+1)/2 nonlinear elements to 3*N-1 linear elements and only N nonlinear element. If f is an invertable increasing function you may even rewrite the last constraint as a simple bound:

```
CAP.LO(I) = f-1(B(I));
```

Some NLP solvers encourage you to move as many nonlinearities as possible into the objective which may make the objective very complex. This is neither recommended nor necessary with CONOPT. A special pre-processing step (discussed in section A4 in Appendix A) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.

6.4 Equalities vs. Inequalities

A resource constraint or a production function is often modeled as an inequality constraint in an optimization model; the optimization algorithm will search over the space of feasible solutions, and if the constraint turns out to constrain the optimal solution the algorithm will make it a binding constraint, and it will be satisfied as an equality. If you know from the economics or physics of the problem that the constraint must be binding in the optimal solution then you have the choice of defining the constraint as an equality from the beginning. The inequality formulation gives a larger feasible space which can make it easier to find a first feasible solution. The feasible space may even be convex. On the other hand, the solution algorithm will have to spend time determining which constraints are binding and which are not. The trade off will therefore depend on the speed of the algorithm component that finds a feasible solution relative to the speed of the algorithm component that determines binding constraints.

In the case of CONOPT, the logic of determining binding constraints is slow compared to other parts of the system, and you should in general make equalities out of all constraints you know must be binding. You can switch to inequalities if CONOPT has trouble finding a first feasible solution.

6.5 Scaling

Nonlinear as well as Linear Programming Algorithms use the derivatives of the objective function and the constraints to determine good search directions, and they 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, determine the relative size of the derivatives and of the function values and thereby also the search path taken by the algorithm.

Assume for example that two goods of equal importance both cost $1 per kg. The first is measured in gram, the second in tons. The coefficients in the cost function will be $1000/g and $0.001/ton, respectively. If cost is measured in $1000 units then the coefficients will be 1 and 1.e-6, and the smaller may be ignored by the algorithm since it is comparable to some of the zero tolerances.

CONOPT assumes implicitly that the model to be solved is well scaled. In this context well scaled means:

- Basic and superbasic solution values are expected to be around 1, e.g. from 0.01 to 100. Nonbasic variables will be at a bound, and the bound values should not be larger than say 100.
- Dual variables (or marginals) on active constraints are expected to be around 1, e.g. from 0.01 to 100. Dual variables on non-binding constraints will of course be zero.
- Derivatives (or Jacobian elements) are expected to be around 1, e.g. from 0.01 to 100.

Variables become well scaled if they are measured in appropriate units. In most cases you should select the unit of measurement for the variables so their expected value is around unity. Of course there will always be some variation. Assume X(I) is the production at location I. In most cases you should select the same unit of measurement for all components of X, for example a value around the average capacity.

Equations become well scaled if the individual terms are measured in appropriate units. After you have selected units for the variables you should select the unit of measurement for the equations so the expected values of the individual terms are around one. If you follow these rules, material balance equations will usually have coefficients of plus and minus one.

Derivatives will usually be well scaled whenever the variables and equations are well scaled. To see if the derivatives are well scaled, run your model with a positive OPTION LIMROW and look for very large or very small coefficients in the equation listing in the GAMS output file.

CONOPT computes a measure of the scaling of the Jacobian, both in the initial and in the final point, and if it seems large it will be printed. The message looks like:

```
** WARNING **  The variance of the derivatives in the initial
               point is large (= 4.1 ). A better initial
               point, a better scaling, or better bounds on the
               variables will probably help the optimization.
```

The variance is computed as SQRT(SUM(LOG(ABS(Jac(i)))**2)/NZ) where Jac(i) represents the NZ nonzero derivatives (Jacobian elements) in the model. A variance of 4.1 corresponds to an average value of LOG(JAC)**2 of 4.1**2, which means that Jacobian values outside the range EXP(-4.1)=0.017 to EXP(+4.1)=60.4 are about as common at values inside. This range is for most models acceptable, while a variance of 5, corresponding to about half the derivatives outside the range EXP(-5)=0.0067 to EXP(+5)=148, can be dangerous.

6.5.1 Scaling of Intermediate Variables

Many models have a set of variables with a real economic or physical interpretation plus a set of intermediate or helping variables that are used to simplify the model. We have seen some of these in section 6.3 on Simple Expressions. It is usually rather easy to select good scaling units for the real variables since we know their order of magnitude from economic or physical considerations. However, the intermediate variables and their defining equations should preferably also be well scaled, even if they do not have an immediate interpretation. Consider the following model fragment where X, Y, and Z are variables and Y is the intermediate variable:

```
SET P / P0*P4 /
PARAMETER A(P) / P0 211, P1 103, P2 42, P3 31, P4 6 /
YDEF .. Y =E= SUM(P, A(P)*POWER(X,ORD(P)-1));
ZDEF .. Z =E= LOG(Y);
```

X lies in the interval 1 to 10 which means that Y will be between 211 and 96441 and Z will be between 5.35 and 11.47. Both X and Z are reasonably scaled while Y and the terms and derivatives in YDEF are about a factor 1.e4 too large. Scaling Y by 1.e4 and renaming it YS gives the following scaled version of the model fragment:

```
YDEFS1 .. YS =E= SUM(P, A(P)*POWER(X,ORD(P)-1))*1.E-4;
ZDEFS1 .. Z  =E= LOG(YS*1.E4);
```

The Z equation can also be written as

```
ZDEFS2 .. Z  =E= LOG(YS) + LOG(1.E4);
```

Note that the scale factor 1.e-4 in the YDEFS1 equation has been placed on the right hand side. The mathematically equivalent equation

```
YDEFS2 .. YS*1.E4 =E= SUM(P, A(P)*POWER(X,ORD(P)-1));
```

will give a well scaled YS, but the right hand side terms of the equation and their derivatives have not changed from the original equation YDEF and they are still far too large.

6.5.2 Using the Scale Option is GAMS

The rules for good scaling mentioned above are exclusively based on algorithmic needs. GAMS has been developed to improve the effectiveness of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as "natural" as possible. The units of measurement is one part of this natural notation, and there is unfortunately often a conflict between what the modeler thinks is a good unit and what constitutes a well scaled model.

To facilitate the translation between a natural model and a well scaled model GAMS has introduced the concept of a scale factor, both for variables and equations. The notation and the definitions are quite simple. First of all, scaling is by default turned off. To turn it on, enter the statement "<model>.SCALEOPT = 1;" in your GAMS program somewhere after the MODEL statement and before the SOLVE statement. "<model>" is the name of the model to be solved. If you want to turn scaling off again, enter the statement "<model>.SCALEOPT = 0;" somewhere before the next SOLVE.

The scale factor of a variable or an equation is referenced with the suffix ".SCALE", i.e. the scale factor of variable X(I) is referenced as X.SCALE(I). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined in assignment statements with X.SCALE(I) on the left hand side,

and scale factors, both from variables and equations, can be used on the right hand side, for example to define other scale factors. The default scale factor is always 1, and a scale factor must be positive; GAMS will generate an execution time error if the scale factor is less than 1.e-20.

The mathematical definition of scale factors is a follows: The scale factor on a variable, V^s is used to related the variable as seen by the modeler, V^m, to the variable as seen by the algorithm, V^a, as follows:

$$V^m = V^a * V^s$$

This means, that if the variable scale, V^s, is chosen to represent the order of magnitude of the modeler's variable, V^m, then the variable seen by the algorithm, V^a, will be around 1.

The scale factor on an equation, G^s, is used to related the equation as seen by the modeler, G^m, to the equation as seen by the algorithm, G^a, as follows:

$$G^m = G^a * G^s$$

This means, that if the equation scale, G^s, is chosen to represent the order of magnitude of the individual terms in the modelers version of the equation, G^m, then the terms seen by the algorithm, G^a, will be around 1.

The derivatives in the scaled model seen by the algorithm, i.e. dG^a/dV^a, are related to the derivatives in the modelers model, dG^m/dV^m, through the formula:

$$dG^a/dV^a = dG^m/dV^m * V^s / G^s$$

i.e. the modelers derivative is multiplied by the scale factor of the variable and divided by the scale factor of the equation. Note, that the derivative is unchanged if $V^s = G^s$. Therefore, if you have a GAMS equation like

```
G .. V =E= expression;
```

and you select $G^s = V^s$ then the derivative of V will remain 1.

If we apply these rules to the example above with an intermediate variable we can get the following automatic scale calculation, based on an "average" reference value for X:

```
SCALAR XREF; XREF = 6;
Y.SCALE = SUM(P, A(P)*POWER(XREF,ORD(P)-1));
YDEF.SCALE = Y.SCALE;
```

or we could scale Y using values at the end of the X interval and add safeguards as follows:

```
Y.SCALE = MAX( ABS(SUM(P, A(P)*POWER(X.LO,ORD(P)-1))),
               ABS(SUM(P, A(P)*POWER(X.UP,ORD(P)-1))),
```

```
                    0.01 );
```

Lower and upper bounds on variables are automatically scaled in the same way as the variable itself. Integer and binary variables cannot be scaled.

GAMS' scaling is in most respects hidden for the modeler. The solution values reported back from a solution algorithm, both primal and dual, are always reported in the user's notation. The algorithm's versions of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if `OPTION LIMROW` and/or `LIMCOL` are positive, and in debugging output from the solution algorithm, generated with `OPTION SYSOUT = ON`. In addition, the numbers in the algorithms iteration log will represent the scaled model: the infeasibilities and reduced gradients will correspond to the scaled model, and if the objective variable is scaled, the value of the objective function will be the scaled value.

A final warning about scaling of multidimensional variables is appropriate. Assume variable X(I,J,K) only appears in the model when the parameter IJK(I,J,K) is nonzero, and assume that CARD(I) = CARD(J) = CARD(K) = 100 while CARD(IJK) is much smaller than 100**2 = 1.e6. Then you should only scale the variables that appear in the model, i.e.

```
X.SCALE(I,J,K)$IJK(I,J,K) = expression;
```

The statement

```
X.SCALE(I,J,K) = expression;
```

will generate records for X in the GAMS database for all combinations of I, J, and K for which the expression is different from 1, i.e. up to 1.e6 records, and apart from spending a lot of time you will very likely run out of memory. Note that this warning also applies to non-default lower and upper bounds.

7. NLP AND DNLP MODELS

GAMS has two classes of nonlinear model, NLP and DNLP. NLP models are defined as models in which all functions that appear with endogenous arguments, i.e. arguments that depend on model variables, are smooth with smooth derivatives. DNLP models can in addition use functions that are smooth but have discontinuous derivatives. The usual arithmetic operators (+, -, *, /, and **) can appear on both model classes.

The functions that can be used with endogenous arguments in a DNLP model and not in an NLP model are ABS, MIN, and MAX and as a consequence the indexed operators SMIN and SMAX.

Note that the offending functions can be applied to expressions that only involve constants such as parameters, var.l, and eq.m. Fixed variables are in principle constants, but GAMS makes its tests based on the functional form of a models, ignoring numerical parameter

values and numerical bound values, and terms involving fixed variables can therefore not be used with ABS, MIN, or MAX in an NLP model.

The NLP solvers used by GAMS can also be applied to DNLP models. However, it is important to know that the NLP solvers attempt to solve the DNLP model as if it was an NLP model. The solver uses the derivatives of the constraints with respect to the variables to guide the search, and it ignores the fact that some of the derivatives may change discontinuously. There are at the moment no GAMS solvers designed specifically for DNLP models and no solvers that take into account the discontinuous nature of the derivatives in a DNLP model.

7.1 DNLP models: what can go wrong?

Solvers for NLP Models are all based on making marginal improvements to some initial solution until some optimality conditions ensure no direction with marginal improvements exist. A point with no marginally improving direction is called a Local Optimum.

The theory about marginal improvements is based on the assumption that the derivatives of the constraints with respect to the variables is a good approximation to the marginal changes in some neighborhood around the current point.

Consider the simple NLP model, min SQR(x), where x is a free variable. The marginal change in the objective is the derivative of SQR(x) with respect to x, which is 2*x. At x = 0, the marginal change in all directions is zero and x = 0 is therefore a Local Optimum.

Next consider the simple DNLP model, min ABS(x), where x again is a free variable. The marginal change in the objective is still the derivative, which is +1 if x > 0 and -1 if x < 0. When x = 0, the derivative depends on whether we are going to increase or decrease x. Internally in the DNLP solver, we cannot be sure whether the derivative at 0 will be -1 or +1; it can depends on rounding tolerances. An NLP solver will start in some initial point, say x = 1, and look at the derivative, here +1. Since the derivative is positive, x is reduced to reduce the objective. After some iterations, x will be zero or very close to zero. The derivative will be +1 or -1, so the solver will try to change x. however, even small changes will not lead to a better objective function. The point x = 0 does not look like a Local Optimum, even though it is a Local Optimum. The result is that the NLP solver will muddle around for some time and then stop with a message saying something like: "The solution cannot be improved, but it does not appear to be optimal."

In this first case we got the optimal solution so we can just ignore the message. However, consider the following simple two-dimensional DNLP model: min ABS(x1+x2) + 5*ABS(x1-x2) with x1 and x2 free variables. Start the optimization from x1 = x2 = 1. Small increases in x1 will increase both terms and small decreases in x1 (by dx) will decrease the first term by dx but it will increase the second term by 5*dx. Any change in x1 only is therefore bad, and it is easy to see that any change in x2 only also is bad. An NLP solver may therefore be stuck in the point x1 = x2 = 1, even though it is not a local solution: the direction (dx1,dx2) = (-1,-1) will lead to the optimum in x1 = x2 = 0. However, the NLP solver cannot distinguish what happens with this model from what

happened in the previous model; the message will be of the same type: "The solution cannot be improved, but it does not appear to be optimal."

7.2 Reformulation from DNLP to NLP

The only reliable way to solve a DNLP model is to reformulate it as an equivalent smooth NLP model. Unfortunately, it may not always be possible. In this section we will give some examples of reformulations.

The standard reformulation approach for the ABS function is to introduce positive and negative deviations as extra variables: The term z = ABS(f(x)) is replaced by z = fplus + fminus, fplus and fminus are declared as positive variables and they are defined with the identity: f(x) =E= fplus - fminus. The discontinuous derivative from the ABS function has disappeared and the part of the model shown here is smooth. The discontinuity has been converted into lower bounds on the new variables, but bounds are handled routinely by any NLP solver. The feasible space is larger than before; f(x) = 5 can be obtained both with fplus = 5, fminus = 0, and z = 5, and with fplus = 1000, fminus = 995, and z = 1995. Provided the objective function has some term that tries to minimize z, either fplus or fminus will become zero and z will end with its proper value.

You may think that adding the smooth constraint fplus * fminus =e= 0 would ensure that either fplus or fminus is zero. However, this type of so-called complementarity constraint is "bad" in any NLP model. The feasible space consists of the two half lines: (fplus = 0 and fminus >= 0) and (fplus >= 0 and fminus=0). Unfortunately, the marginal change methods used by most NLP solvers cannot move from one half line to the other, and the solution is stuck at the half line it happens to reach first.

There is also a standard reformulation approach for the MAX function. The equation z =E= MAX(f(x),g(y)) is replace by the two inequalities, z =G= f(x) and z =G= g(y). Provided the objective function has some term that tries to minimize z, one of the constraints will become binding as equality and z will indeed be the maximum of the two terms.

The reformulation for the MIN function is similar. The equation z =E= MIN(f(x),g(y)) is replaced by the two inequalities, z =L= f(x) and z =L= g(y). Provided the objective function has some term that tries to maximize z, one of the constraints will become binding as equality and z is indeed the minimum of the two terms.

MAX and MIN can have more than two arguments and the extension should be obvious.

The non-smooth indexed operators, SMAX and SMIN can be handled using a similar technique: for example, z =E= SMAX(I, f(x,I)) is replaced by the indexed inequality: Ineq(I) .. z =L= f(x,I);

The reformulations that are suggested here all enlarge the feasible space. They require the objective function to move the final solution to the intersection of this larger space with the original feasible space. Unfortunately, the objective function is not always so helpful. If it is not, you may try using one of the smooth approximations described next. However, you should realize, that if the objective function cannot help the "good" approximations

described here, then your overall model is definitely non-convex and it is likely to have multiple local optima.

7.3 Smooth Approximations

Smooth approximations to the non-smooth functions ABS, MAX, and MIN are approximations that have function values close to the original functions, but have smooth derivatives.

A smooth GAMS approximation for ABS(f(x)) is

```
SQRT( SQR(f(x)) + SQR(delta) )
```

where delta is a small scalar. The value of delta can be used to control the accuracy of the approximation and the curvature around f(x) = 0. The approximation error is largest when f(x) is zero, in which case the error is delta. The error is reduced to approximately SQR(delta)/2 for f(x) = 1. The second derivative is 1/delta at f(x) = 0 (excluding terms related to the second derivative of f(x)). A delta value between 1.e-3 and 1.e-4 should in most cases be appropriate. It is possible to use a larger value in an initial optimization, reduce it and solve the model again. You should note, that if you reduce delta below 1.d-4 then large second order terms might lead to slow convergence or even prevent convergence.

The approximation shown above has its largest error when f(x) = 0 and smaller errors when f(x) is far from zero. If it is important to get accurate values of ABS exactly when f(x) = 0, then you may use the alternative approximation

```
SQRT( SQR(f(x)) + SQR(delta) ) – delta
```

instead. The only difference is the constant term. The error is zero when f(x) is zero and the error grows to -delta when f(x) is far from zero.

Some theoretical work uses the Huber, H(*), function as an approximation for ABS. The Huber function is defined as

```
H(x) = x for x > delta,
H(x) = -x for x < -delta and
H(x) = SQR(x)/2/delta + delta/2 for –delta < x < delta.
```

Although the Huber function has some nice properties, it is for example accurate when ABS(x) > delta, it is not so useful for GAMS work because it is defined with different formulae for the three pieces.

A smooth GAMS approximation for MAX(f(x),g(y)) is

```
( f(x) + g(y) + SQRT( SQR(f(x)-g(y)) + SQR(delta) ) )/2
```

where delta again is a small scalar. The approximation error is delta/2 when f(x) = g(y) and decreases with the difference between the two terms. As before, you may subtract a constant term to shift the approximation error from the area f(x) = g(y) to areas where the difference is large. The resulting approximation becomes

```
( f(x) + g(y) + SQRT( SQR(f(x)-g(y)) + SQR(delta) ) – delta )/2
```

Similar smooth GAMS approximations for MIN(f(x),g(y)) are

```
( f(x) + g(y) - SQRT( SQR(f(x)-g(y)) + SQR(delta) ) )/2
```

and

```
( f(x) + g(y) - SQRT( SQR(f(x)-g(y)) + SQR(delta) ) + delta )/2
```

Appropriate delta values are the same as for the ABS approximation: in the range from 1.e-2 to 1.e-4

It appears that there are no simple symmetric extensions for MAX and MIN of three or more arguments or for indexed SMAX and SMIN.

7.4 Are DNLP models always non-smooth?

A DNLP model is defined as a model that has an equation with an ABS, MAX, or MIN function with endogenous arguments. The non-smooth properties of DNLP models are derived from the non-smooth properties of these functions through the use of the chain rule. However, composite expressions involving ABS, MAX, or MIN can in some cases have smooth derivatives and the model can therefore in some cases be smooth.

One example of a smooth expression involving an ABS function is common in water systems modeling. The pressure loss over a pipe, dH, is proportional to the flow, Q, to some power, P. P is usually around +2. The sign of the loss depend on the direction of the flow so dH is positive if Q is positive and negative if Q is negative. Although GAMS has a SIGN function, it cannot be used in a model because of its discontinuous nature. Instead, the pressure loss can be modeled with the equation dH =E= const * Q * ABS(Q)**(P-1), where the sign of the Q-term takes care of the sign of dH, and the ABS function guaranties that the real power ** is applied to a non-negative number. Although the expression involves the ABS function, the derivatives are smooth as long as P is greater than 1. The derivative with respect to Q is const * (P-1) * ABS(Q)**(P-1) for Q > 0 and -const * (P-1) * ABS(Q)**(P-1) for Q < 0. The limit for Q going to zero from both right and left is 0, so the derivative is smooth in the critical point Q = 0 and the overall model is therefore smooth.

Another example of a smooth expression is the following terribly looking Sigmoid expression:

```
Sigmoid(x) = exp( min(x,0) ) / (1+exp(-abs(x)))
```

The standard definition of the sigmoid function is

```
Sigmoid(x) = exp(x) / ( 1+exp(x) )
```

This definition is well behaved for negative and small positive x, but it not well behaved for large positive x since exp overflows. The alternative definition:

```
Sigmoid(x) = 1 / ( 1+exp(-x) )
```

is well behaved for positive and slightly negative x, but it overflows for very negative x. Ideally, we would like to select the first expression when x is negative and the second when x is positive, i.e.

```
Sigmoid(x) = (exp(x)/(1+exp(x)))$(x lt 0) + (1/(1+exp(-x)))$(x gt 0)
```

but a $-control that depends on an endogenous variable is illegal. The first expression above solves this problem. When x is negative, the nominator becomes exp(x) and the denominator becomes 1+exp(x). And when x is positive, the nominator becomes exp(0) = 1 and the denominator becomes 1+exp(-x). Since the two expressions are mathematically identical, the combined expression is of course smooth, and the exp function is never evaluated for a positive argument.

Unfortunately, GAMS cannot recognize this and similar special cases so you must always solve models with endogenous ABS, MAX, or MIN as DNLP models, even in the cases where the model is smooth.

7.5 Are NLP models always smooth?

NLP models are defined as models in which all operators and functions are smooth. The derivatives of composite functions, that can be derived using the chain rule, will therefore in general be smooth. However, it is not always the case. The following simple composite function is not smooth: y = SQRT(SQR(x)). The composite function is equivalent to y = ABS(x), one of the non-smooth DNLP functions.

What went wrong? The chain rule for computing derivatives of a composite function assumes that all intermediate expressions are well defined. However, the derivative of SQRT grows without bound when the argument approaches zero, violating the assumption.

There are not many cases that can lead to non-smooth composite functions, and they are all related to the case above: The real power, x**y, for 0 < y < 1 and x approaching zero. The SQRT function is a special case since it is equivalent to x**y for y = 0.5.

If you have expressions involving a real power with an exponent between 0 and 1 or a SQRT, you should in most cases add bounds to your variables to ensure that the derivative or any intermediate terms used in their calculation become undefined. In the example above, SQRT(SQR(x)), a bound on x is not possible since x should be allowed to be both positive and negative. Instead, changing the expression to SQRT(SQR(x) + SQR(delta)) may lead to an appropriate smooth formulation.

Again, GAMS cannot recognize the potential danger in an expression involving a real power, and the presence of a real power operator is not considered enough to flag a model as a DNLP model. During the solution process, the NLP solver will compute constraint values and derivatives in various points within the bounds defined by the modeler. If these calculations result in undefined intermediate or final values, a function evaluation error is reported, an error counter is incremented, and the point is flagged as a bad point. The following action will then depend on the solver. The solver may try to continue, but only if the modeler has allowed it with an "Option Domlim = xxx". The problem of detecting discontinuities is changed from a structural test at the GAMS model generation stage to a dynamic test during the solution process.

You may have a perfectly nice model in which intermediate terms become undefined. The composite function SQRT(POWER(x,3)) is mathematically well defined around x = 0, but the computation will involve the derivative of SQRT at zero, that is undefined. It is the modeler's responsibility to write expressions in a way that avoids undefined intermediate terms in the function and derivatives computations. In this case, you may either add a small strictly positive lower bound on x or rewrite the function as x**1.5.

APPENDIX A: ALGORITHMIC INFORMATION

The objective of this Appendix is to give technically oriented users some understanding of what CONOPT is doing so they can get more information out of the iteration log. This information can be used to prevent or circumvent algorithmic difficulties or to make informed guesses about which options to experiment with to improve CONOPT's performance on particular model classes.

A1. Overview of GAMS/CONOPT

GAMS/CONOPT is a GRG-based algorithm specifically designed for large nonlinear programming problems expressed in the following form

```
min or max   f(x)          (1)
subject to   g(x) = b      (2)
lo ≤ x ≤ up                (3)
```

where `x` is the vector of optimization variables, `lo` and `up` are vectors of lower and upper bounds, some of which may be minus or plus infinity, `b` is a vector of right hand sides, and `f` and `g` are differentiable nonlinear functions that define the model. n will in the following

denote the number of variables and m the number of equations. (2) will be referred to as the (general) constraints and (3) as the bounds.

The relationship between the mathematical model in (1)-(3) above and the GAMS model is simple: The inequalities defined in GAMS with `=L=` or `=G=` are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all GAMS equalities to ensure that the Jacobian matrix, i.e. the matrix of derivatives of the functions `g` with respect to the variables `x`, has full row rank. All these slacks are together with the normal GAMS variables included in `x`. `lo` represent the lower bounds defined in GAMS, either implicitly with the `POSITIVE VARIABLE` declaration, or explicitly with the `VAR.LO` notation, as well as any bounds on the slacks. Similarly, `up` represent upper bounds defined in GAMS, e.g. with the `VAR.UP` notation, as well as any bounds on the slacks. `g` represent the non-constant terms of the GAMS equations themselves; non-constant terms appearing on the right hand side are by GAMS moved to the left hand side and constant terms on the left hand side are moved to the right. The objective function `f` is simply the GAMS variable to be minimized or maximized.

Additional comments on assumptions and design criteria can be found in the Introduction to the main text.

A2. The CONOPT Algorithm

The algorithm used in GAMS/CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details on the algorithm can be found in Drud (1985 and 1992). Here we will just give a short verbal description of the major steps in a generic GRG algorithm. The later sections in this Appendix will discuss some of the enhancements in CONOPT that make it possible to solve large models.

The key steps in any GRG algorithm are:

1. Initialize and Find a feasible solution.
2. Compute the Jacobian of the constraints, J.
3. Select a set of n basic variables, x_b, such that B, the sub-matrix of basic column from J, is nonsingular. Factorize B. The remaining variables, x_n, are called nonbasic.
4. Solve $B^T u = df/dx_b$ for the multipliers u.
5. Compute the reduced gradient, $r = df/dx - J^T u$. r will by definition be zero for the basic variables.
6. If r projected on the bounds is small, then stop. The current point is close to optimal.
7. Select the set of superbasic variables, x_s, as a subset of the nonbasic variables that profitably can be changed, and find a search direction, d_s, for the superbasic variables based on r_s and possibly on some second order information.
8. Perform a line search along the direction d. For each step, x_s is changed in the direction d_s and x_b is subsequently adjusted to

```
      satisfy g(x_b,x_s) = b in a pseudo-Newton process using the factored
      B from step 3.
   9. Go to 2.
```

The individual steps are of course much more detailed in a practical implementation like CONOPT. Step 1 consists of several pre-processing steps as well as a special Phase 0 and a scaling procedure as described in the following sections A3 to A6. The optimizing steps are specialized in several versions according to the whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the matrices J and B can be avoided or done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 can be improved by observing that the optimal step as in LP almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized: the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this (section A7 and A8). If the model is "very" linear an improved search direction (step 7) can be computed using some specialized inner LP-like iterations (section A9), and a steepest edge procedure can be useful for certain models that needs very many iterations (section A10).

The remaining two sections give some short guidelines for selecting non-default options (section A11), and discuss miscellaneous topics (section A12) such as CONOPT's facilities for strictly triangular models (A12.1) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear Systems (A12.2), as well as numerical difficulties due to loss of feasibility (A12.3) and slow or no progress due to stalling (A12.4).

A3. Iteration 0: The Initial Point

The first few "iterations" in the iteration log (see section 2 in the main text for an example) are special initialization iterations, but they have been counted as real iterations to allow the user to interrupt at various stages during initialization. Iteration 0 corresponds to the input point exactly as it was received from GAMS. The sum of infeasibilities in the column labeled "Infeasibility" includes all residuals, also from the objective constraint where "Z =E= expression" will give rise to the term abs(Z - expression) that may be nonzero if Z has not been initialized. You may stop CONOPT after iteration 0 with `"OPTION ITERLIM = 0;"` in GAMS. The solution returned to GAMS will contain the input point and the values of the constraints in this point. The marginals of both variables and equations have not yet been computed and they will be returned as EPS.

This possibility can be used for debugging when you have a reference point that should be feasible, but is infeasible for unknown reasons. Initialize all variables to their reference values, also all intermediated variables, and call CONOPT with `ITERLIM = 0`. Then compute and display the following measures of infeasibility for each block of constraints, represented by the generic name EQ:

```
=E= constraints: ROUND(ABS(EQ.L - EQ.LO),3)
=L= constraints: ROUND(MIN(0,EQ.L - EQ.UP),3)
=G= constraints: ROUND(MIN(0,EQ.LO - EQ.L),3)
```

The ROUND function rounds to 3 decimal places so GAMS will only displays the infeasibilities that are larger than 5.e-4.

Similar information can be derived from inspection of the equation listing generated by GAMS with `"OPTION LIMROW = nn;"`, but although the method of going via CONOPT requires a little more work during implementation it can be convenient in many cases, for example for large models and for automated model checking.

A4. Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraints-variable pairs that can be solved a priori (so-called pre-triangular equations and variables) are solved and the corresponding variables are assigned their final values. Constraints that always can be made feasible because they contain a free variable with a constant coefficient (so-called post-triangular equation-variable pairs) are excluded from the search for a feasible solution, and from the Infeasibility measure in the iteration log. Implicitly, the equations and variables are ordered as shown in Fig. 1.

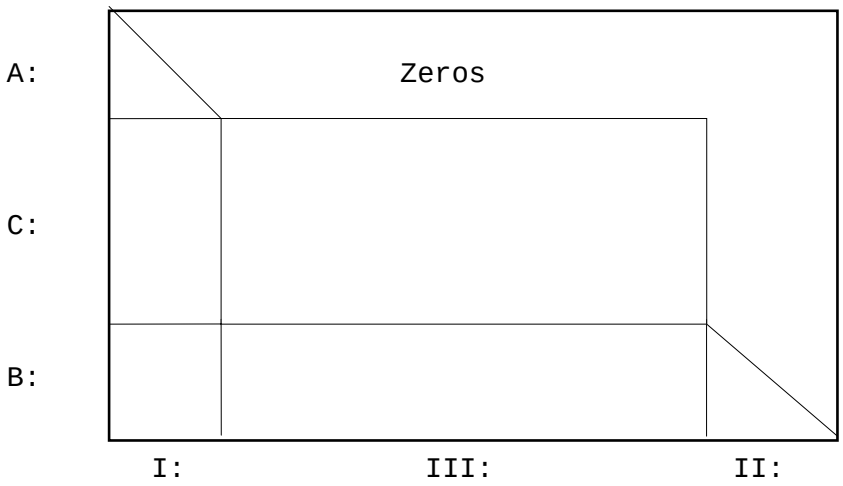

Figure 1: The ordered Jacobian after Preprocessing.

A4.1 Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Fig. 1. They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, GAMS/CONOPT looks for equations with only one non-fixed variable. If such an equation exists, GAMS/CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section A12.1.

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```
VARIABLE X1, X2, X3, OBJ;
EQUATION E1, E2, E3;
E1 .. LOG(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
X1.LO = 0.1;
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;
```

Equation `E2` is first solved with respect to `X2` (result 3/5 = 0.6). It is easy to solve the equation since `X2` appears linearly, and the result will be unique. `X2` is then fixed and the equation is removed. Equation `E1` is now a candidate since `X1` is the only remaining non-fixed variable in the equation. Here `X1` appears nonlinearly and the value of `X1` is found using an iterative scheme based on Newton's method. The iterations are started from the value provided by the modeler or from the default initial value. In this case `X1` is started from the default initial value, i.e. the lower bound of 0.1, and the result after some iterations is `X1` = 2.718 = EXP(1).

During the recursive solution process it may not be possible to solve one of the equations. If the lower bound on `X1` in the model above is changed to 3.0 you will get the following output:

```
** An equation in the pre-triangular part of the model cannot
   be solved because the critical variable is at a bound.
```

```
Residual=           9.86122887E-02
Tolerance (RTNWTR)= 6.34931126E-07

E1: Infeasibility in pre-triangular part of model.
X1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and
variables is:

E2 is solved with respect to
X2. Solution value =  6.0000000000E-01

E1 could not be solved with respect to
X1. Final solution value =  3.0000000000E+00
E1 remains infeasible with residual = 9.8612288668E-02
```

The problem is as indicated that the variable to be solved for is at a bound, and the value suggested by Newton's method is on the infeasible side of the bound. The critical variable is `X1` and the critical equation is `E1`, i.e. `X1` tries to exceed its bound when CONOPT solves equation `E1` with respect to `X1`. To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility: In this case equation `E2` was first solved with respect to variable `X2`, then equation `E1` was attempted to be solved with respect to `X1` at which stage the problem appeared. To make the analysis easier CONOPT will always report the minimal set of equations and variables that caused the infeasibility.

Another type of infeasibility is shown by the following model:

```
VARIABLE X1, X2, X3, OBJ;
EQUATION E1, E2, E3;
E1 .. SQR(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;
```

where `LOG(X1)` has been replaced by `SQR(X1)` and the lower bound on `X1` has been removed. This model gives the message:

```
** An equation in the pre-triangular part of the model cannot
   be solved because of too small pivot.
   Adding a bound or initial value may help.

   Residual=           4.0000000
   Tolerance (RTNWTR)= 6.34931126E-07

   E1: Infeasibility in pre-triangular part of model.
   X1: Infeasibility in pre-triangular part of model.
```

```
   The solution order of the critical equations and
   variables is:

   E2 is solved with respect to
   X2. Solution value =  6.0000000000E-01

   E1 could not be solved with respect to
   X1. Final solution value =  0.0000000000E+00
   E1 remains infeasible with residual =-4.0000000000E+00
```

After equation E2 has been solved with respect to X2, equation E1 that contains the term $X1^2$ should be solved with respect to X1. The initial value of X1 is the default value zero. The derivative of E1 with respect to X1 is therefore zero, and it is not possible for CONOPT to determine whether to increase or decrease X1. If X1 is given a nonzero initial value the model will solve. If X1 is given a positive initial value the equation will give X1 = 1, and if X1 is given a negative initial value the equation will give X1 = -1.

The last type of infeasibility that can be detected during the solution of the pre-triangular or recursive equations is shown by the following example

```
VARIABLE X1, X2, X3, OBJ;
EQUATION E1, E2, E3, E4;
E1 .. LOG(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
E4 .. X1 + X2 =E= 3.318;
X1.LO = 0.1;
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;
```

that is derived from the first model by the addition of equation E4. This model produces the following output

```
** An equation is inconsistent with other equations in the
   pre-triangular part of the model.

   Residual=           2.81828458E-04
   Tolerance (RTNWTR)= 6.34931126E-07

   The pre-triangular feasibility tolerance may be relaxed with
   a line:

         SET       RTNWTR    X.XX

   in the CONOPT control program.

   E4: Inconsistency in pre-triangular part of model.

   The solution order of the critical equations and
   variables is:

   E2 is solved with respect to
   X2. Solution value =  6.0000000000E-01

   E1 is solved with respect to
   X1. Solution value =  2.7182818285E+00
```

```
All variables in equation E4 are now fixed
and the equation is infeasible. Residual = 2.8182845830E-04
```

First E2 is solved with respect to X2, then E1 is solved with respect to X1 as indicated by the last part of the output. At this point all variables that appear in equation E4, namely X1 and X2, are fixed, but the equation is not feasible. E4 is therefore inconsistent with E1 and E2 as indicated by the first part of the output. In this case the inconsistency is fairly small, 2.8E-04, so it could be a tolerance problem. CONOPT will always report the tolerance that was used, rtnwtr - the triangular Newton tolerance, and if the infeasibility is small it will also tell how the tolerance can be relaxed. Section 5 in the main text on "The CONOPT Options File" gives further details on how to change tolerances, and a complete list of options is given in Appendix B.

You can turn the identification and solution of pre-triangular variables and equations off by adding the line "lspret = f" in the CONOPT control program. This can be useful in some special cases where the point defined by the pre-triangular equations gives a function evaluation error in the remaining equations. The following example shows this:

```
VARIABLE X1, X2, X3, X4, OBJ;
EQUATION E1, E2, E3, E4;
E1 .. LOG(1+X1) + X2 =E= 0;
E2 .. 5 * X2 =E= -3;
E3 .. OBJ =E= 1*SQR(X1) + 2*SQRT(0.01 + X2 - X4) + 3*SQR(X3);
E4 .. X4 =L= X2;
MODEL FER / ALL /; SOLVE FER4 MINIMIZING OBJ USING NLP;
```

All the nonlinear functions are defined in the initial point in which all variables have their default value of zero. The pre-processor will compute X2 = -0.6 from E2 and X1 = 0.822 from E1. When CONOPT continues and attempts to evaluate E3, the argument to the SQRT function is negative when these new triangular values are used together with the initial X4 = 0, and CONOPT cannot backtrack to some safe point since the function evaluation error appears the first time E3 is evaluated. When the pre-triangular preprocessor is turned off, X2 and X4 are changed at the same time and the argument to the SQRT function remains positive throughout the computations. Note, that although the purpose of the E4 inequality is to guarantee that the argument of the SQRT function is positive in all points, and although E4 is satisfied in the initial point, it is not satisfied after the pre-triangular constraints have been solved. Only simple bounds are strictly enforced at all times. Also note that if the option "lspret = f" is used then feasible linear constraints will in fact remain feasible.

An alternative (and preferable) way of avoiding the function evaluation error is to define an intermediate variable equal to 0.01+X2-X4 and add a lower bound of 0.01 on this variable. The inequality E4 could then be removed and the overall model would have the same number of constraints.

A4.2 Preprocessing: Post-triangular Variables and Constraints

Consider the following fragment of a larger GAMS model:

```
VARIABLE UTIL(T)  Utility in period T
         TOTUTIL  Total Utility;
EQUATION UTILDEF(T) Definition of Utility
         TUTILDEF   Definition of Total Utility;
UTILDEF(T).. UTIL(T) =E= nonlinear function of other variables;
TUTILDEF  .. TOTUTIL =E= SUM( T , UTIL(T) / (1+R)**ORD(T) );
MODEL DEMO / ALL /; SOLVE DEMO MAXIMIZING TOTUTIL USING NLP;
```

The part of the model shown here is easy to read and from a modeling point of view it should be considered well written. However, it could be more difficult to solve than a model in which variable `UTIL(T)` was substituted out because all the `UTILDEF` equations are nonlinear constraints that the algorithms must ensure are satisfied.

To make well written models like this easy to solve CONOPT will move as many nonlinearities as possible from the constraints to the objective function. This automatically changes the model from the form that is preferable for the modeler to the form that is preferable for the algorithm. In this process CONOPT looks for free variables that only appear in one equation outside the objective function. If such a variable exists and it appears linearly in the equation, like `UTIL(T)` appears with coefficient 1 in equation `UTILDEF(T)`, then the equation can always be solved with respect to the variable. This means that the variable logically can be substituted out of the model and the equation can be removed. The result is a model that has one variable and one equation less, and a more complex objective function. As variables and equations are substituted out, new candidates for elimination may emerge, so CONOPT repeats the process until no more candidates exist.

This so-called post-triangular preprocessing step will often move several nonlinear constraints into the objective function where they are much easier to handle, and the effective size of the model will decrease. In some cases the result can even be a model without any general constraints. The name post-triangular is derived from the way the equations and variables appear in the permuted Jacobian in fig. 1. The post-triangular equations and variables are the ones on the lower right hand corner labeled B and II, respectively.

In the example above, the `UTIL` variables will be substituted out of the model together with the nonlinear `UTILDEF` equations provided the `UTIL` variables are free and do not appear elsewhere in the model. The resulting model will have fewer nonlinear constraints, but more nonlinear terms in the objective function.

Although you may know that the nonlinear functions on the right hand side of `UTILDEF` always will produce positive `UTIL` values, you should in general not declare `UTIL` to be a `POSITIVE VARIABLE`. If you do, GAMS/CONOPT may not be able to eliminate `UTIL(T)`, and the model will be harder to solve. It is of course unfortunate that a redundant bound changes the solution behavior, and to reduce this problem the new

CONOPT2 will try to estimate the range of nonlinear expressions using interval arithmetic. If the computed range of the right hand side of the UTILDEF constraint is within the bounds of UTIL, then these bounds cannot be binding and UTIL is a so-called implied free variable that can be eliminated.

The following model fragment from a least squares model shows another case where the preprocessing step in GAMS/CONOPT is useful:

```
VARIABLE RESIDUAL(CASE)  Residuals
         SSQ             Sum of Squared Residuals;
EQUATION EQEST(CASE)     Equation to be estimated
         SSQDEF          Definition of objective;
EQEST(CASE).. RESIDUAL(CASE) =E= expression in other variables;
SSQDEF     .. SSQ =E= SUM( CASE, SQR( RESIDUAL(CASE) ) );
MODEL LSQLARGE / ALL /; SOLVE LSQLARGE USING NLP MINIMIZING SSQ;
```

GAMS/CONOPT will substitute the RESIDUAL variables out of the model using the EQEST equations. The model solved by GAMS/CONOPT is therefore mathematically equivalent to the following GAMS model

```
VARIABLE SSQ     Sum of Squared Residuals;
EQUATION SSQD    Definition of objective;
SSQD .. SSQ =E= SUM( CASE, SQR(expression in other variables));
MODEL LSQSMALL / ALL /;
SOLVE LSQSMALL USING NLP MINIMIZING SSQ;
```

However, if the "expression in other variables" is a little complicated, e.g. if it depends on several variables, then the first model, LSQLARGE, will be much faster to generate with GAMS because its derivatives in equation EQEST and SSQDEF are much simpler than the derivatives in the combined SSQD equation in the second model, LSQSMALL. The larger model will therefore be faster to generate, and it will also be faster to solve because the derivative computations will be faster.

Note that the comments about what are good model formulations are dependent on the preprocessing capabilities in GAMS/CONOPT. Other algorithms may prefer models like LSQSMALL over LSQLARGE. Also note that the variables and equations that are substituted out are still indirectly part of the model. GAMS/CONOPT evaluates the equations and computes values for the variables each time the value of the objective function is needed, and their values are available in the GAMS solution.

The number of pre- and post-triangular equations and variables is printed in the log file between iteration 0 and 1 as shown in the iteration log in Section 2 of the main text. The sum of infeasibilities will usually decrease from iteration 0 to 1 because fewer constraints usually will be infeasible. However, it may increase as shown by the following example:

```
POSITIVE VARIABLE X, Y, Z;
EQUATION E1, E2;
E1.. X =E= 1;
E2.. 10*X - Y + Z =E= 0;
```

started from the default values X.L = 0, Y.L = 0, and Z.L = 0. The initial sum of infeasibilities is 1 (from E1 only). During pre-processing X is selected as a pre-triangular variable in equation E1 and it is assigned its final value 1 so E1 becomes feasible. After this change the sum of infeasibilities increases to 10 (from E2 only).

You may stop CONOPT after iteration 1 with "OPTION ITERLIM = 1;" in GAMS. The solution returned to GAMS will contain the pre-processed values for the variables that can be assigned values from the pre-triangular equations, the computed values for the variables used to solve the post-triangular equations, and the input values for all other variables. The pre- and post-triangular constraints will be feasible, and the remaining constraints will have values that correspond to this point. The marginals of both variables and equations have not been computed yet and will be returned as EPS.

The crash procedure described in the following sub-section is an optional part of iteration 1.

A4.3 Preprocessing: The Optional Crash Procedure

In the initial point given to CONOPT the variables are usually split into a group with initial value provided by the modeler (in the following called the assigned variables) and a group of variables for which no initial value has been provided (in the following called the default variables). The objective of the optional crash procedure is to find a point in which as many of the constraints as possible are feasible, primarily by assigning values to the default variables and by keeping the assigned variables at their initial values. The implicit assumption in this procedure is that if the modeler has assigned an initial value to a variable then this value is "better" then a default initial value.

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a singleton row appears, or until all variables have been fixed. The variables to be fixed at their initial value are selected using a heuristic that both tries to create many row singletons and tries to select variables with "good values". Since the values of many variables will come to depend in the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.

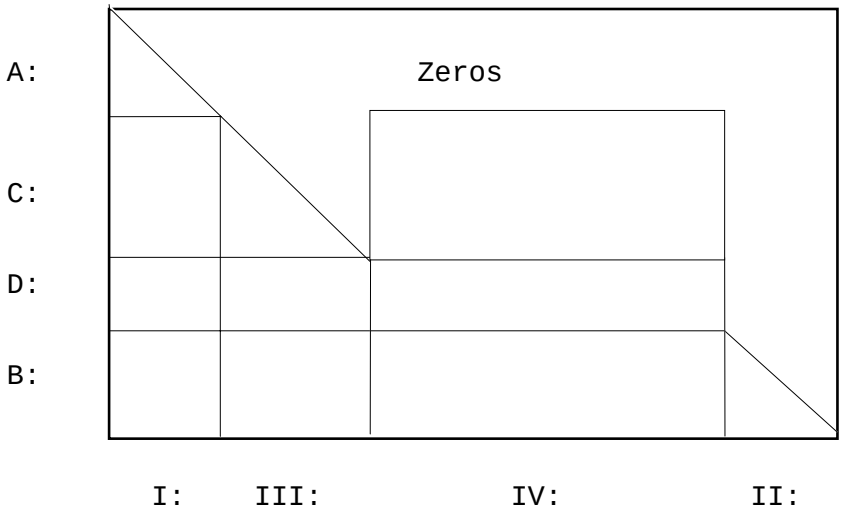

Figure 2: The ordered Jacobian after Preprocessing and Crashing.

Fig. 2 shows a reordered version of fig. 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g. intermediate variables. The number of crash-triangular variables is shown on the iteration output between iteration 0 and 1, but only if the crash procedure is turned on.

The result of the crash procedure is an updated initial point in which usually a large number of equations will be feasible, namely all equations labeled A, B, and C in Fig. 2. There is, as already shown with the small example in section A4.2 above, no guarantie that the sum of infeasibilities will be reduced, but it is often the case, and the point will often provide a good starting point for the following procedures that finds an initial feasible solution.

The crash procedure is activated by adding the line `"lstcrs=t"` in the options file. The default value of `lstcrs` (`lstcrs` = Logical Switch for Triangular CRaSh) is `f` or `false`, i.e. the crash procedure is not normally used.

The Crash procedure is only available in CONOPT2.

A5. Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The Infeasibility column shows the scaled sum of infeasibilities. You may again stop

CONOPT after iteration 2 with "OPTION ITERLIM = 2;" in GAMS, but the solution that is reported in GAMS will have been scaled back again so there will be no change from iteration 1 to iteration 2.

The following description of the automatic scaling procedure is included for completeness. Experiments have so far not given the results we hoped for, and scaling is therefore by default turned off, corresponding to the CONOPT option "lsscal = f". Users are recommended to be cautious with the automatic scaling procedure. If scaling is a problem, try to use manual scaling or scaling in GAMS (see section 6.5 in the main text) based on an understanding of the model.

The scaling procedure multiplies all variables in group III and all constraints in group C (see Fig. 1) by scale factors. All derivatives (Jacobian elements) are scaled by a row and a column factors, and the objective of the scaling procedure is (1) to minimize the deviation of the absolute values of the scaled Jacobian elements from one, and (2) to scale very large variables towards one in absolute value. The scaling procedure is based on simple equilibration: the rows of the Jacobian are divided by the geometic mean of the largest and smallest Jacobian element, the columns are then divided by the geometic mean of the largest and smallest Jacobian element (adjusted if the scaled variable is very large), and the process is repeated until the changes are small.

The main difficulties involved in automatic scaling for NLP models are how to take account of

1. very rapidly varying Jacobian element, e.g. elements associated with LOG(X) or 1/X as X approaches a very small lower bound, and
2. very small Jacobian elements, e.g. elements associated with product terms in which one of the factors is close to zero.

CONOPT tries to overcome these difficulties by (1) rescaling at regular intervals (see lfscal) and (2) by rounding very small elements up (rtminj) and by disallowing very large and very small scale factors (rtmins and rtmaxs).

The CR-Cells that control scaling, lsscal, lfscal, rtminj, rtmins, and rtmaxs, are all described in Appendix B.

The Scaling procedure is only available in CONOPT2.

A6. Finding a Feasible Solution: Phase 0

The GRG algorithm used by CONOPT is a feasible path algorithm. This means that once it has found a feasible point it tries to remain feasible and follow a path of improving feasible points until it reaches a local optimum. CONOPT starts with the point provided by GAMS. This point will always satisfy the bounds (3): GAMS will simply move a variable that is outside its bounds to the nearer bound before it is presented to the solver. If the general constraints (2) also are feasible then CONOPT will work with feasible solutions throughout the optimization. However, the initial point may not satisfy the general

constraints (2). If this is not the case, GAMS/CONOPT must first find an initial feasible point. This first step can be just as hard as finding an optimum for some models. For some models feasibility is the only problem.

GAMS/CONOPT has two methods for finding an initial feasible point. The first method is not very reliable but it is fast when it works; the second method is reliable but slower. The fast method is called Phase 0 and it is described in this section. It is used first. The reliable method, called Phase 1 and 2, will be used if Phase 0 terminates without a feasible solution.

Phase 0 is based on the observation that Newton's method for solving a set of equations usually is very fast, but it may not always converge. Newton's method in its pure form is defined for a model with the same number of variables as equations, and no bounds on the variables. With our type of model there are usually too many variables, i.e. too many degrees of freedom, and there are bounds. To get around the problem of too many variables, GAMS/CONOPT selects a subset with exactly m "basic" variables to be changed. The rest of the variables will remain fixed at their current values, that are not necessarily at bounds. To accommodate the bounds, GAMS/CONOPT will try to select variables that are away from their bounds as basic, subject to the requirement that the Basis matrix, consisting of the corresponding columns in the Jacobian, must have full rank and be well conditioned.

The Newton equations are solved to yield a vector of proposed changes for the basic variables. If the full proposed step can be applied we can hope for the fast convergence of Newton's method. However, several things may go wrong:

a) The infeasibilities, measured by the 1-norm of g (i.e. the sum of the absolute infeasibilities, excluding the pre- and post-triangular equations), may not decrease as expected due to nonlinearities.

b) The maximum step length may have to be reduced if a basic variable otherwise would exceed one of its bounds.

In case a) GAMS/CONOPT tries various heuristics to find a more appropriate set of basic variables. If this does not work, some "difficult" equations, i.e. equations with large infeasibilities and significant nonlinearities, are temporarily removed from the model, and Newton's method is applied to the remaining set of "easy" equations.

In case b) GAMS/CONOPT will remove the basic variable that first reaches one of its bounds from the basis and replace it by one of the nonbasic variables. Newton's method is then applied to the new set of basic variables. The logic is very close to that of the dual simplex method. In cases where some of the basic variables are exactly at a bound GAMS/CONOPT uses an anti degeneracy procedure based on Ryan and Osborne (1988) to prevent cycling.

Phase 0 will end when all equations except possibly some "difficult" equations are feasible within some small tolerance. If there are no difficult equations, GAMS/CONOPT has found a feasible solution and it will proceed with Phase 3 and 4. Otherwise, Phase 1 and 2 is used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: Iter, Phase, Ninf, Infeasibility, Step, MX, and OK. The number in the Ninf column counts the number of "difficult" infeasible equations, and the number in the Infeasibility column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. There are three possible combinations of values in the MX and OK columns: combination (1) has F in the MX column and T in the OK column and it will always be combined with 1.0 in the Step column: this is an ideal Newton step. The infeasibilities in the easy equations should be reduced quickly, but the difficult equations may dominate the number in the Infeasibility column so you may not observe it. However, a few of these iterations is usually enough to terminate Phase 0. Combination (2) has T in the MX column indicating that a basic variable has reached its bound and is removed from the basis as in case b) above. This will always be combined with T in the OK column. The Step column will show a step length less than the ideal Newton step of 1.0. Combination (3) has F in both the MX and OK column. It is the bad case and will always be combined with a step of 0.0: this is an iteration where nonlinearities are dominating and one of the heuristics from case a) must be used.

The success of the Phase 0 procedure is based on being able to choose a good basis that will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so GAMS/CONOPT has some variables away from their bounds to select from. This topic was discussed in more detail in section 6.1 on "Initial Values".

The start and the iterations of Phase 0 can, in addition to the crash option described in section A6, be controlled with the three CR-cells `lslack, lsmxbs,` and `lmmxsf` described in Appendix B.

A7. Finding a Feasible Solution: Phase 1 and 2

Most of the equations will be feasible when phase 0 stops. To remove the remaining infeasibilities CONOPT uses a procedure similar to the phase 1 procedure used in Linear Programming: artificial variables are added to the infeasible equations (the equations with Large Residuals), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variable are already part of the model as slack variables; their bounds are simply relaxed temporarily.

This infeasibility minimization problem is similar to the overall optimization problem: minimize an objective function subject to equality constraints and bounds on the variables. The feasibility problem is therefore solved with the ordinary GRG optimization procedure. As the artificial variables gradually become zero, i.e. as the infeasible equations become feasible, they are taken out of the auxiliary objective function. The number of infeasibilities (shown in the Ninf column of the log file) and the sum of infeasibilities (in the Infeasibility column) will therefore both decrease monotonically.

The iteration output will label these iterations as phase 1 and/or phase 2. The distinction between phase 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phase 3 and 4 that is described in the next sections.

A8. Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in A2 above. The factorization in step 3 is performed using an efficient sparse LU factorization similar to the one described by Suhl and Suhl (1990). The matrix operations in step 4 and 5 are also performed sparse.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model is locally. When the model appears to be fairly linear in the area in which the optimization is performed, i.e. when the function and constraint values are close to their linear approximation for the steps that are taken, then CONOPT takes advantages of the linearity: The derivatives (the Jacobian) are not computed in every iteration, the basis factorization is updated using cheap LP techniques as described by Reid (1982), the search direction is determined without use of second order information, i.e. similar to a steepest descend algorithm, and the initial steplength is estimated as the step length where the first variable reaches a bound; very often, this is the only step length that has to be evaluated. These cheap almost linear iterations are refered to a Linear Mode and they are labeled Phase 1 when the model is infeasible and objective is the sum of infeasibilities and Phase 3 when the model is feasible and the real objective function is optimized.

When the constraints and/or the objective appear to be more nonlinear CONOPT will still follow step 2 to 9 of the GRG algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed in each iteration since the nonlinearities imply that the derivatives change. On the other hand, the set of basic variables will often be the same and CONOPT will take advantage of this during the factorization of the basis. In step 7 CONOPT uses the BFGS algorithm to estimate second order information and determine search directions. And in step 8 it will often be necessary to perform more than one step in the line search. These nonlinear iterations are labeled Phase 2 in the output if the solution is still infeasible, and Phase 4 if it is feasible. The iterations in phase 2 and 4 are in general more expensive than the iteration in phase 1 and 3.

Some models will remain in phase 1 (linear mode) until a feasible solution is found and then continue in phase 3 until the optimum is found, even if the model is truly nonlinear. However, most nonlinear models will have some iterations in phase 2 and/or 4 (nonlinear mode). Phase 2 and 4 indicates that the model has significant nonlinear terms around the current point: the objective or the constraints deviate significantly from a linear model for the steps that are taken. To improve the rate of convergence CONOPT tries to estimate second order information in the form of an estimated reduced Hessian using the BFGS formula.

Each iteration is, in addition to the step length shown in column "Step", characterized by two logicals: MX and OK. MX = T means that the step was maximal, i.e. it was determined by a variable reaching a bound. This is the expected value in Phase 1 and 3. MX = F means that no variable reached a bound and the optimal step length will in general be determined by nonlinearities. OK = T means that the line search was well-behaved and an optimal step length was found; OK = F means that the line search was ill-behaved, which means that CONOPT would like to take a larger step, but the feasibility restoring Newton process used during the line search did not converge for large step lengths. Iterations marked with OK = F (and therefore also with MX = F) will usually be expensive, while iterations marked with MX = T and OK = T will be cheap.

A9. Linear Mode: The SLP Procedure

When the model continues to appear linear CONOPT will often take many small steps, each determined by a new variable reaching a bound. Although the line searches are fast in linear mode, each require one or more evaluations of the nonlinear constraints, and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraint evaluations CONOPT may replace the steepest descend direction in step 7 of the GRG algorithm with a sequential linear programming (SLP) technique to find a search direction that anticipates the bounds on all variables and therefore gives a larger expected change in objective in each line search. The search direction and the last basis from the SLP procedure are used in an ordinary GRG-type line search in which the solution is made feasible at each step. The SLP procedure is only used to generate good directions; the usual feasibility preserving steps in CONOPT are maintained, so CONOPT is still a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SLP-mode are identified by numbers in the column labeled "SlpIt" in the iteration log. The number in the SlpIt column is the number of non-degenerate SLP iterations. This number is adjusted dynamically according to the success of the previous iterations and the perceived linearity of the model.

The SLP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model. This is indicated with MX = T and OK = T.
- The objective is nonlinear along the search direction and the optimal step is less than one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SLP procedure or not, based on progress information. You may turn it off completely with the line `"lseslp = f"` in the CONOPT options file (usually `conopt2.opt`). The default value of `lseslp` (`lseslp` = Logical Switch Enabling SLP mode) is `t` or `true`, i.e. the SLP procedure is enabled and

CONOPT may use it when considered appropriate. It is seldom necessary to define `lseslp`, but it can be useful if CONOPT repeatedly turns SLP on and off, i.e. if you see a mixture of lines in the iteration log with and without numbers in the SlpIt column.

The SLP procedure is only available in CONOPT2.

A10. Linear Mode: The Steepest Edge Procedure

When optimizing in linear mode (Phase 1 or 3) CONOPT will by default use a steepest descend algorithm to determine the search direction. CONOPT allows you to use a Steepest Edge Algorithm as an alternative. The idea, borrowed from Linear Programming, is to scale the nonbasic variables according to the Euclidean norm of the "updated column" in a standard LP tableau, the so-called edge length. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables which has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length, and the large change in basic variables is more likely to give rise to larger nonlinear terms.

The steepest edge algorithm has been very successful for linear programs, and our initial experience has also shown that it will give fewer iterations for most nonlinear models. However, the cost of maintaining the edge lengths can be more expensive in the nonlinear case and it depends on the model whether steepest edge results in faster overall solution times or not. CONOPT uses the updating methods for the edge lengths from LP, but it must re-initialize the edge lengths more frequently, e.g. when an inversion fails, which happens more frequently in nonlinear models than in linear models, especially in models with many product terms, e.g. blending models, where the rank of the Jacobian can change from point to point.

Steepest edge is turned on with the line, `"lsanrm = t"`, in the CONOPT options file (usually conopt2.opt). The default value of `lsanrm` (`lsanrm` = Logical Switch for A-NoRM) is `f` or `false`, i.e. the steepest edge procedure is turned off.

The steepest edge procedure is mainly useful during linear mode iterations. However, it has some influence in phase 2 and 4 also: The estimated reduced Hessian in the BFGS method is initialized to a diagonal matrix with elements on the diagonal computed from the edge lengths, instead of the usual scaled unit matrix.

The Steepest Edge procedure is only available in CONOPT2.

A11. How to Select Non-default Options

The non-default options have an influence on different phases of the optimization and you must therefore first observe whether most of the time is spend in Phase 0, Phase 1 and 3, or in Phase 2 and 4.

Phase 0: The quality of Phase 0 depends on the number of iterations and on the number and sum of infeasibilities after Phase 0. The iterations in Phase 0 are much faster than the other iterations, but the overall time spend in Phase 0 may still be rather large. If this is the case, or if the infeasibilities after Phase 0 are large you may try to use the triangular crash options:

```
lstcrs = t
```

Observe if the initial sum of infeasibility after iteration 1 has been reduced, and if the number of phase 0 iterations and the number of infeasibilities at the start of phase 1 have been reduced. If `lstcrs` reduces the initial sum of infeasibilities but the number of iterations still is large you may try:

```
lslack = t
```

CONOPT will after the preprocessor immediately add artificial variables to all infeasible constraints so Phase 0 will be eliminated, but the sum and number of infeasibilities at the start of Phase 1 will be larger. You are in reality trading Phase 0 iterations for Phase 1 iterations.

You may also try the experimental bending line search with

```
lmmxsf = 1
```

The line search in Phase 0 will with this option be different and the infeasibilities may be reduced faster than with the default `"lmmxsf = 0"`. It is likely to be better if the number of iterations with both MX = F and OK = F is large. This option may be combined with `"lstcrs = t"`. Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

Phase 1 and 3: The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, `"lsanrm = t"`, but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but work in progress tries to push this limit upwards. Try it when the number of iterations is very large, or when many iterations are poorly behaved identified with OK = F in the iteration log.

The default SLP mode is usually an advantage, but it is too expensive for a few models. If you observe frequent changes between SLP mode and non-SLP mode, or if many line searches in the SLP iterations are ill-behaved with OK = F, then it may be better to turn SLP off with `"lseslp = f"`.

Phase 2 and 4: There are currently not many options available if most of the time is spend in Phase 2 and Phase 4. If the change in objective during the last iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, `rtredg`. If the number of superbasic variables is larger than 500 then it may be worth while to increase the maximum size of the estimated Hessian matrix,

`lfnsup`. Note that you in this case may need more memory than CONOPT allocates by default.

A12: Miscellaneous Topics

A12.1 Triangular Models

A triangular model is one in which the non-fixed variables and the equations can be sorted such that the first equation only depends on the first variable, the second equation only depends on the first two variables, and the p-th equation only depends on the first p variables. Provided there are no difficulties with bounds or small pivots, triangular models can be solved one equation at a time using the method describe in section "A4.1 Preprocessing: Pre-triangular Variables and Constraints" and the solution process will be very fast and reliable.

Triangular models can in many cases be useful for finding a good initial feasible solution: Fix a subset of the variables so the remaining model is known to be triangular and solve this triangular simulation model. Then reset the bounds on the fixed variables to their original values and solve the original model. The first solve will be very fast and if the fixed variables have been fixed at good values then the solution will also be good. The second solve will start from the good feasible solution generated by the first solve and it will usually optimize much more quickly than from a poor start.

The modeler can instruct CONOPT that a model is supposed to be triangular with the option "`lstria = t`". CONOPT will then use a special version of the preprocessing routine (see section 4.1) that solves the model very efficiently. If the model is solved successfully then CONOPT terminates with the message:

```
** Feasible solution to a recursive model.
```

and the Model Status will be 2, Locally Optimal, or 1, Optimal, depending on whether there were any nonlinear pivots or not. All marginals on both variables and equations are returned as 0 (zero) or EPS.

Two SOLVEs with different option files can be arranged by writing the option files as they are needed from within the GAMS program with `PUT` statements followed by a `PUTCLOSE`. You can also have two different option files, for example `conopt2.opt` and `conopt2.op2`, and select the second with the GAMS statement "`<model>.optfile = 2;`".

The triangular facility handles a number of error situations:

1. Non-triangular models: CONOPT will ensure that the model is indeed triangular. If it is not, CONOPT will return model status 5, Locally Infeasible, plus some information that allows the modeler to identify the mistake. The necessary information is related to the order of the variables and equations and number of occurrences of variables and equations,

and since GAMS does no have a natural place for this type of information CONOPT returns it in the marginals of the equations and variables. The solution order for the triangular equations and variables that have been solved successfully are defined with positive numbers in the marginals of the equations and variables. For the remaining non-triangular variables and equations CONOPT shows the number of places they appear as negative numbers, i.e. a negative marginal for an equation shows how many of the non-triangular variables that appear in this equation. You must fix one or more variables until at least one of the non-triangular equation only has one non-fixed variable left.

2. Infeasibilities due to bounds: If some of the triangular equations cannot be solved with respect to their variable because the variable will exceed the bounds, then CONOPT will flag the equation as infeasible, keep the variable at the bound, and continue the triangular solve. The solution to the triangular model will therefore satisfy all bounds and almost all equations. The termination message will be

```
** Infeasible solution. xx artificial(s) have been
   introduced into the recursive equations.
```

and the model status will be 5, Locally Infeasible.

The modeler may in this case add explicit artificial variables with high costs to the infeasible constraints and the resulting point will be an initial feasible point to the overall optimization model. You will often from the mathematics of the model know that only some of the constraints can be infeasible, so you will only need to check whether to add artificials in these equations. Assume that a block of equations MATBAL(M,T) could become infeasible. Then the artificials that may be needed in this equation can be modeled and identified automatically with the following GAMS constructs:

```
SET APOSART(M,T) Add a positive artificial in Matbal
    ANEGART(M,T) Add a negative artificial in Matbal;
APOSART(M,T) = NO; ANEGART(M,T) = NO;

POSITIVE VARIABLE
    VPOSART(M,T) Positive artificial variable in Matbal
    VNEGART(M,T) Negative artificial variable in Matbal;

MATBAL(M,T).. Left hand side =E= right hand side
   + VPOSART(M,T)$APOSART(M,T) - VNEGART(M,T)$ANEGART(M,T);

OBJDEF.. OBJ =E= other_terms +
         WEIGHT * SUM((M,T), VPOSART(M,T)$APOSART(M,T)
                           +VNEGART(M,T)$ANEGART(M,T) );

Solve triangular model ...

APOSART(M,T)$(MATBAL.L(M,T) GT MATBAL.UP(M,T)) = YES;
ANEGART(M,T)$(MATBAL.L(M,T) LT MATBAL.LO(M,T)) = YES;

Solve final model ...
```

3. Small pivots: The triangular facility requires the solution of each equation to be locally unique which also means that the pivots used to solve each equation must be nonzero. The model segment

```
E1 .. X1 =E= 0;
E2 .. X1 * X2 =E= 0;
```

will give the message

```
   X2 appearing in
   E2: Pivot too small for triangular model. Value=0.000E+00

** Infeasible solution. The equations were assumed to be
   recursive but they are not. A pivot element is too small.
```

However, the uniqueness of `X2` may not be relevant if the solution just is going to be used as an initial point for a second model. The option `"lsismp = t"` (for Logical Switch: Ignore SMall Pivots) will allow zero pivots as long as the corresponding equation is feasible for the given initial values.

A12.2 Constrained Nonlinear System or Square Systems of Equations

There is a special model class in GAMS called `CNS` - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e. a model in which the number of non-fixed variables is equal to the number of constraints. Currently, CONOPT2 and PATHCNS are the only solvers for this model class. A CNS model can be solved with a solve statement like

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

without an objective term. In some cases it may be convenient to solve aCNS model with a standard solve statement combined with an options file that has the statement `"lssqrs = t"`. In the latter case, CONOPT2 will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT2 will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. The solution process will stop with an error message and the current intermediate infeasible solution will be returned if the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

The solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is

```
** Feasible solution to a square system.
```

that usually will be combined with model status 16 –Solved. If CONOPT2 in special cases can guarantie that the solution is unique, for example if the model is linear, then the model status will be 15 –Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints

```
e1 ..   x1 +   x2 =e= 1;
e2 .. 2*x1 + 2*x2 =e= 2;
```

will result in the message

```
** Error in Square System: Pivot too small.
    e2: Pivot too small.
    x1: Pivot too small.
```

"Pivot too small" means that the set of constraints is linearly dependent and there cannot be a unique solution to the model. The message points to one variable and one constraint. However, this just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error will usually be combined with model status 5 – Locally Infeasible. In the cases where CONOPT2 can guarantie that the infeasibility is not caused by nonlinearities the model status will be 4 –Infeasible. If the constraints are linearly dependent but the current point satisfy the constraints then the solution status will be 17 –Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

A model with these two constraints and the bound

```
e1 .. x1 + x2 =e= 2;
e2 .. x1 - x2 =e= 0;
x1.lo = 1.5;
```

will result in the message

```
** Error in Square System: A variable tries to exceed its bound.
    x1: The variable tries to exceed its bound.
```

because the solution, (x1,x2) = (1,1) violates the bound on x1. This error case also be combined with model status 5 – Locally Infeasible. In the cases where CONOPT2 can guarantie that the infeasibility is not caused by nonlinearities the model status will be 4 – Infeasible. If you encounter problems with active bounds but you think it is caused by

nonlinearities and that there is a solution, then you may try to use the bending linesearch with option "`lmmxsf = t`".

The CNS facility can be used to generate an initial feasible solution in almost the same way as the triangular model facility: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the `CNS` solver or with `lssqrs = t`, reset the bounds on the fixed variables, and solve the original model. The CNS facility can be used on a larger class of models that include simultaneous sets of equations. However, the square system must be non-singular and feasible; CONOPT cannot, like in the triangular case, add artificial variables to some of the constraints and solve the remaining system when a variable reaches one of its bounds.

Additional information on CNS model can be found at GAMS web site at the address http://www.gams.com/docs/pdf/cns.pdf

---

A12.3 Loss of Feasibility

During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message "Loss of Feasibility - Return to Phase 0". The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization where the objective changes rapidly fairly large infeasibilities may be acceptable. As the change in objective in each iteration becomes smaller it will be necessary to solve the constraints more accurately so the "noise" in objective value from the inaccurate constraints will remain smaller than the real change. The noise is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g. when an inequality becomes binding. In these cases CONOPT will tighten the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick and it happens silently. However, Newton's method may fail, for example in cases where the model is degenerate and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section A6. Finding a Feasible Solution: Phase 0. and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the "Loss of feasibility ..." message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.

In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. We

are working on reducing this problem. Until a final solution has been implemented you are encouraged to (1) consider if bounds on some degenerate variables can be removed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, `rtnwma` and `rtnwmi` (see Appendix B), if this happens with your model.

A12.4 Stalling

CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly.

The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus a noise adjustment term equal to the scalar product of the residuals with the marginals (see section A12.3 where this noise term also is used). The noise adjustment term is very useful in allowing CONOPT to work smoothly with fairly inaccurate intermediate solutions. However, there is a disadvantage: the noise adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT looses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To avoid infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see OK = F in section A8). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds `lfstal` and `lfstal` is positive. The default value of `lfstal` is 100. The message will be:

```
** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum fairly accurately then you may have to increase the value of `lfstal`.

A12.5 External Functions

Both CONOPT and CONOPT2 can be used with external functions written in a programming language such as Fortran or C. Additional information is available at GAMS's web site at http://www.gams.com/docs/extfunc.htm.

APPENDIX B - CR-CELLS

The CR-Cells that ordinary GAMS users can access are listed below. CR-Cells starting on R assume real values, CR-Cells starting on LS assume logical values (TRUE, T, FALSE, or F), and all other CR-Cells starting on L assume integer values. Several CR-Cells are only used in CONOPT2 in which case it will be mentioned below. However, these CR-Cells can still be defined in an options file for the old CONOPT, but they will silently be ignored:

`lfilog` Iteration Log frequency. A log line is printed to the screen every `lfilog` iterations (see also `lfilos`). The default value depends on the size of the model: it is 10 for models with less than 500 constraints, 5 for models between 501 and 2000 constraints and 1 for larger models. The log itself can be turned on and off with the Logoption (LO) parameter on the GAMS call.

`Lfilos` Iteration Log frequency for SLP iterations. A log line is printed to the screen every `lfilos` iterations while using the SLP mode. The default value depends on the size of the model: it is 5 for models with less than 500 constraints, and 1 for larger models. Only CONOPT2.

`lfmxns` Limit on new superbasics. When there has been a sufficient reduction in the reduced gradient in one subspace, CONOPT tests if any nonbasic variables should be made superbasic. The ones with largest reduced gradient of proper sign are selected, up to a limit of `lfmxns`. The default value of `lfmxns` is 5. The limit is replaced by the square root of the number of structural variables if `lfmxns` is set to zero.

`lfnicr` Limit for slow progress / no increase. The optimization is stopped with a "Slow Progress" message if the change in objective is less than 10 * `rtobjr` * max(1,abs(FOBJ)) for `lfnicr` consecutive iterations where FOBJ is the value of the current objective function. The default value of `lfnicr` is 12.

`lfnsup` Maximum Hessian dimension. If the number of superbasics exceeds `lfnsup` CONOPT will no longer use the BFGS algorithm but will switch to a steepest descend approach, independent of the degree of nonlinearity of the model. The default value of `lfnsup` is 500. If `lfnsup` is increased beyond its default value the default memory allocation may not be sufficient, and you may have to include a "<model>.WORKSPACE = xx.x;" statement in your GAMS source file, where "model" represent the GAMS name of the model. You should try to increase the value of `lfnsup` if CONOPT performs many iterations in Phase 4 with the number of superbasics (NSB) larger than `lfnsup` and without much progress. The new value should in this case be larger than the number of superbasics.

`lfscal` Frequency for scaling. The scale factors are recomputed after `lfscal` recomputations of the Jacobian. The default value is 20. Only CONOPT2

`lfstal` Maximum number of stalled iterations. If `lfstal` is positive then CONOPT will stop with a "No change in objective" message when the number of stalled iterations as defined in section A12.4 exceeds `lfstal` and `lfstal` is positive. The default value of `lfstal` is 100. Only CONOPT2.

`lmmxsf` Method for finding the maximal step while searching for a feasible solution. The step in the Newton direction is usually either the ideal step of 1.0 or the step that will bring the first basic variable exactly to its bound. An alternative procedure uses "bending": All variables are moved a step s and the variables that are outside their bounds after this step are projected back to the bound. The step length is determined as the step where the sum of infeasibilities, computed from a linear approximation model, starts to increase again. The advantage of this method is that it often can make larger steps and therefore better reductions in the sum of infeasibilities, and it is not very sensitive to degeneracies. The alternative method is turned on by setting `lmmxsf` to 1, and it is turned off by setting `lmmxsf` to 0. Until the method has received additional testing it is by default turned off. Only CONOPT2.

`lsismp` Logical switch for Ignoring Small Pivots. `Lsismp` is only used when `lstria = t`. If `lsismp = t` (default is `f` or `false`) then a triangular equations is accepted even if the pivot is almost zero (less than `rtpivt` for nonlinear elements and less than `rtpiva` for linear elements), provided the equation is feasible, i.e. with residual less than `rtnwtr`. Only CONOPT2.

`lslack` Logical switch for slack basis. If `lslack = t` then the first basis after preprocessing will have slacks in all infeasible constraints and Phase 0 will usually be bypassed. This is sometimes useful together with `lstcrs = t` if the number of infeasible constraints after the crash procedure is small. This is especially true if the SLP procedure described in section A9 quickly can remove these remaining infeasibilities. It is necessary to experiment with the model to determine if this option is useful.

`lsmxbs` Logical Switch for Maximal Basis. `lsmxbs` determines whether CONOPT should try to improve the condition number of the initial basis (`t` or `true`) before starting the Phase 0 iterations or just use the initial basis immediately (`f` or `false`). The default value is `t`, i.e. CONOPT tries to improve the basis. There is a computational cost associated with the procedure, but it will usually be saved because the better conditioning will give rise to fewer Phase 0 iterations and often also to fewer large residuals at the end of Phase 0. The option is ignored if `lslack` is `true`. Only CONOPT2.

`lspost` Logical switch for the Post-triangular preprocessor. If `lspost = f` (default is `t` or `true`) then the post-triangular preprocessor discussed in section A4.2 is turned off.

`lspret` Logical switch for the Pre-triangular preprocessor. If `lspret = f` (default is `t` or `true`) then the pre-triangular preprocessor discussed in section A4.1 is

turned off.

`lsscal` Logical switch for scaling. A logical switch that turns scaling on (with the value `t` or `true`) or off (with the value `f` or `false`). The default value is `f`, i.e. no scaling. Only CONOPT2.

`lssqrs` Logical switch for Square Systems. If `lssqrs = t` (default is `f` or `false`), then the model must be a square system as discussed in section A12.2. Users are recommended to use the CNS model class in GAMS. Only CONOPT2.

`lstria` Logical switch for triangular models. If `lstria = t` (default is `f` or `false`) then the model must be triangular as discussed in section A12.1. Only CONOPT2.

`rtmaxj` Maximum Jacobian element. The optimization is stopped if a Jacobian element exceeds this value. `rtmaxj` is initialized to a value that depends on the machine precision. It is on most machines around 2.d5. The actual value is shown by CONOPT in connection with "Too large Jacobian element" messages. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve.

`rtmaxv` Internal value of infinity. The model is considered unbounded if a variable exceeds `rtmaxv` in absolute value. `rtmaxv` is initialized to a value that depends on the machine precision. It is on most machines around 6.d7. The actual value is shown by CONOPT in connection with "Unbounded" messages. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve.

`rtmaxs` Scale factors larger than `rtmaxs` are rounded down to `rtmaxs`. The default value is 1024. Only CONOPT2.

`rtminj` All Jacobian elements with a value less than `rtminj` are rounded up to the value `rtminj` before scaling is started to avoid problems with zero and very small Jacobian elements. The default value is 1.E-5. Only CONOPT2.

`rtmins` Scale factors smaller than `rtmins` are rounded up to `rtmins`. The default value is 1/1024. (All scale factors are powers of 2 to avoid round-off errors from the scaling procedure). Only CONOPT2.

`rtnwma` Maximum feasibility tolerance. A constraint will only be considered feasible if the residual is less than `rtnwma` times MaxJac, independent on the dual variable. MaxJac is an overall scaling measure for the constraints computed as max(1,maximal Jacobian element/100). The default value of `rtnwma` is 1.e-3.

`rtnwmi` Minimum feasibility tolerance. A constraint will always be considered feasible if the residual is less than `rtnwmi` times MaxJac (see above), independent of the dual variable. The default value depends on the machine precision. It is on most machines around 4.d-10. You should only increase this number if you have inaccurate function values and you get an infeasible solution with a very small sum of infeasibility, or if you have very large terms in some of your

constraints (in which case scaling may be more appropriate). Square systems (see `lssqrs` and section A12.2) are always solved to the tolerance `rtnwmi`.

`rtnwtr` Triangular feasibility tolerance. If you solve a model, fix some of the variables at their optimal value and solve again and the model then is reported infeasible in the pre-triangular part, then you should increase `rtnwtr`. The infeasibilities in some unimportant constraints in the "Optimal" solution have been larger than `rtnwtr`. The default value depends on the machine precision. It is on most machines around 6.d-7.

`rtobjr` Relative objective tolerance. CONOPT assumes that the reduced objective function can be computed to an accuracy of `rtobjr` * max(1,abs(FOBJ)) where FOBJ is the value of the current objective function. The default value of `rtobjr` is machine specific. It is on most machines around 3.d-13. The value is used in tests for "Slow Progress", see `lfnicr`.

`rtoned` Relative accuracy of one-dimensional search. The one-dimensional search is stopped if the expected further decrease in objective estimated from a quadratic approximation is less than `rtoned` times the decrease obtained so far. The default value is 0.2. A smaller value will result in more accurate but more expensive line searches and this may result in an overall decrease in the number of iterations. Values above 0.7 or below 0.01 should not be used.

`rtpiva` Absolute pivot tolerance. A pivot element is only considered acceptable if its absolute value is larger than `rtpiva`. The default value is 1.d-10. You may have to decrease this value towards 1.d-11 or 1.d-12 on poorly scaled models.

`rtpivr` Relative pivot tolerance. A pivot element is only considered acceptable relative to other elements in the column if its absolute value is at least `rtpivr` * the largest absolute value in the column. The default value is 0.05. You may have to increase this value towards one on poorly scaled models. Increasing `rtpivr` will result in denser L and U factors of the basis.

`rtpivt` Triangular pivot tolerance. A nonlinear triangular pivot element is considered acceptable if its absolute value is larger than `rtpivt`. The default value is 1.e-7. Linear triangular pivot must be larger than `rtpiva`.

`rtredg` Optimality tolerance. The reduced gradient is considered zero and the solution optimal if the largest superbasic component is less than `rtredg`. The default value depends on the machine, but is usually around 9.d-8. If you have problems with slow progress or stalling you may increase `rtredg`. This is especially relevant for very large models.

`rvspac` A space allocation factor that sometime can speed up the solution of square systems. CONOPT will tell you if it is worth while to set this parameter to a non-default value for your class of model.

`rvstlm` Step length multiplier. The step length in the one-dimensional line search is not allowed to increased by a factor of more than `rvstlm` between steps for models with nonlinear constraints and a factor of 100 * `rvstlm` for models

with linear constraints. The default value is 4.

APPENDIX C: REFERENCES

J. Abadie and J. Carpentier, Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints, in *Optimization*, R. Fletcher (ed.), Academic Press, New York, 37-47 (1969).

A. Drud, A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems, *Mathematical Programming 31*, 153-191 (1985).

A.S. Drud, CONOPT - A Large-Scale GRG Code, *ORSA Journal on Computing 6*, 207-216 (1992).

A.S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Tutorial for CONOPT Subroutine Library, 16p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark, 1995.

A.S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Reference Manual for CONOPT Subroutine Library, 69p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark, 1996.

J.K. Reid, A Sparsity Exploiting Variant of Bartels-Golub Decomposition for Linear Programming Bases, *Mathematical Programming 24*, 55-69 (1982).

D.M. Ryan and M.R. Osborne, On the Solution of Highly Degenerate Linear Programmes, *Mathematical Programming 41*, 385-392 (1988).

U.H. Suhl and L.M. Suhl, Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, *ORSA Journal on Computing 2*, 325-335 (1990).

GAMS/Cplex 7.5 User Notes

Table of Contents

---

1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention. Access is provided (subject to proper licensing) to Cplex solution algorithms including the Linear Optimizer and the Barrier and Mixed Integer Solvers. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

---

2 How to run a model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```
Option LP = Cplex ;   { or RMIP or MIP }
```

The above statement should appear before the Solve statement. The MIP capability is separately licensed, so you may not be able to use Cplex for MIP problems on your system. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

If you have purchased the parallel option, it can be used by specifying

```
Option LP = CplexPar ;   { or RMIP or MIP }
```

or by specifying CplexPar as the default solver during GAMS installation. Parallel GAMS/Cplex also requires installation of an ILOG license file before it will work. Contact support@gams.com for instructions.

---

3 Overview of Cplex

3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex's state of the art modified primal simplex algorithm. Certain types of problems benefit from using the alternative dual simplex algorithm, the network optimizer, or the barrier algorithm.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled Physical Memory Limitations carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the dual simplex algorithm rather than the default primal simplex algorithm. In particular, highly degenerate problems with little variability in the right-hand-side coefficients but significant variability in the cost coefficients often solve much faster using dual simplex. Also, very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying dual simplex if numerical problems occur while using primal simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has exactly 2 nonzero entries, one with a +1 coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

GAMS/Cplex also provides access to the Cplex Infeasibility Finder. The Infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing.

3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and bound algorithm (with cuts) which solves a series of LP subproblems. Because a single mixed integer problem generates many LP subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

---

4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

OPTION Bratio = x;	Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.
`Option IterLim = n;`	Sets the iteration limit. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. For MIP problems, controlling the length of the solution run by limiting the execution time (ResLim) is preferable. IterLim is not used at all when solving an LP with the barrier algorithm. By default, Cplex will use an iteration limit based on problem characteristics. To specify a barrier iteration limit, use Cplex parameter baritlim.
`OPTION ResLim = x;`	Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution.
`Option SysOut = On;`	Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.
`ModelName``.Cheat = x;`	Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option objdif overrides the GAMS cheat parameter.
`ModelName``.Cutoff = x;`	Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.
`ModelName``.NodLim = x;`	Maximum number of nodes to process for a MIP problem.
`ModelName``.OptCA = x;`	Absolute optimality criterion for a MIP problem.

ModelName.OptCR = x;	Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when (|BP - BF|)/(1.0e-10 + |BF|) < OptCR where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is: (|BP - BF|)/(|BP|) < OptCR
ModelName.OptFile = 1;	Instructs Cplex to read the option file. The name of the option file is *cplex.opt*.
ModelName.PriorOpt = 1;	Instructs Cplex to use priority branching information passed by GAMS through the *variable*.prior parameters.
ModelName.TryInt = x;	Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. Moving the value to the bound allows the variable to be part of an initial integer solution when using parameter mipstart. The preferred branching direction will only be effective when priorities are used.

---

5 Summary of Cplex Options

The various Cplex options are listed here by category, with a few words about each to indicate its function. The options are listed again, in alphabetical order and with detailed descriptions, in the last section of this document.

5.1 Preprocessing and General Options

advind	advanced basis use
aggfill	aggregator fill parameter
aggind	aggregator on/off
coeredind	coefficient reduction on/off
depind	dependency checker on/off
interactive	allow interactive option setting after a Control-C
names	load GAMS names into Cplex
objrng	do objective ranging
precompress	presolve compression
predual	give dual problem to the optimizer
preind	turn presolver on/off
prepass	number of presolve applications to perform
printoptions	write values of all options to the GAMS listing file
reduce	primal and dual reduction type
relaxpreind	presolve for initial relaxation on/off
rerun	rerun problem if presolve infeasible or unbounded
rhsrng	do right-hand-side ranging
rngrestart	write GAMS readable ranging information file
scaind	matrix scaling on/off
tilim	overrides the GAMS ResLim option
workdir	directory for working files
workmem	memory available for working storage

5.2 Simplex Algorithmic Options

craind	crash strategy (used to obtain starting basis)
dpriind	dual simplex pricing
epper	perturbation constant
iis	run the IIS finder if the problem is infeasible
iisind	IIS finder method to use
lpmethod	algorithm to be used for LP problems
netfind	attempt network extraction
netppriind	network simplex pricing
perind	force initial perturbation
perlim	number of stalled iterations before perturbation
ppriind	primal simplex pricing
pricelim	pricing candidate list
reinv	refactorization frequency
simthreads	number of threads for parallel dual simplex algorithm

5.3 Simplex Limit Options

itlim	iteration limit
netitlim	iteration limit for network simplex
objllim	objective function lower limit
objulim	objective function upper limit
singlim	limit on singularity repairs

5.4 Simplex Tolerance Options

epmrk	Markowitz pivot tolerance
epopt	optimality tolerance
eprhs	feasibility tolerance
netepopt	optimality tolerance for the network simplex method
neteprhs	feasibility tolerance for the network simplex method

5.5 Barrier Specific Options

baralg	algorithm selection
barcolnz	dense column handling
barcrossalg	barrier crossover method
barepcomp	convergence tolerance
bargrowth	unbounded face detection
baritlim	iteration limit
barmaxcor	maximum correction limit
barobjrng	maximum objective function

barooc	out-of-core barrier
barorder	row ordering algorithm selection
barthreads	number of threads for parallel barrier algorithm
barvarup	variable upper limit

5.6 MIP Algorithmic Options

bbinterval	best bound interval
bndstrenind	bound strengthening
brdir	set branching direction
bttol	backtracking limit
cliques	clique cut generation
covers	cover cut generation
cutlo	lower cutoff for tree search
cuts	default cut generation
cutsfactor	cut limit
cutup	upper cutoff for tree search
disjcuts	disjunctive cuts generation
flowcovers	flow cover cut generation
flowpaths	flow path cut generation
fraccuts	Gomory fractional cut generation
gubcovers	GUB cover cut generation
heurfreq	heuristic frequency
implbd	implied bound cut generation
mipemphasis	MIP solution tactics
mipordind	priority list on/off
mipordtype	priority order generation
mipstart	use mip starting values
mipthreads	number of threads for parallel mip algorithm
mircuts	mixed integer rounding cut generation
nodefileind	node storage file indicator
nodelim	maximum number of nodes to process
nodesel	node selection strategy
preslvnd	node presolve selector
probe	perform probing before solving a MIP
startalg	algorithm for initial LP
strongcandlim	size of the candidates list for strong branching
strongitlim	limit on iterations per branch for strong branching
strongthreadlim	number of threads for strong branching
subalg	algorithm for subproblems
varsel	variable selection strategy at each node

5.7 MIP Limit Options

aggcutlim	aggrigation limit for cut generation
cutpass	maximum number of cutting plane passes
fraccand	candidate limit for generating Gomory fractional cuts
fracpass	maximum number of passes for generating Gomory fractional cuts
intsollim	maximum number of integer solutions
nodelim	maximum number of nodes to solve
trelim	maximum space in memory for tree

5.8 MIP Tolerance Options

epagap	absolute stopping tolerance
epgap	relative stopping tolerance
epint	integrality tolerance
objdif	over-rides GAMS Cheat parameter
relobjdif	relative cheat parameter

5.9 Output Options

bardisplay	progress display level
mipdisplay	progress display level
mipinterval	progress display interval
netdisplay	network display level
quality	write solution quality statistics
simdisplay	simplex display level
writebas	produce a Cplex basis file
writelp	produce a Cplex LP file
writemps	produce a Cplex MPS file
writeord	produce a Cplex ord file
writesav	produce a Cplex binary problem file
writesos	produce a Cplex sos file

5.10 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs). Anything after the value will be ignored.

Following is an example options file *cplex.opt.*

```
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

6 Special Notes

6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically.

Cplex performs an operation called refactorization at a frequency determined by the reinv option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorizing is an expensive operation, increasing the refactorization frequency by reducing the reinv option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare SOS1 and SOS2 variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

> A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

> A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.

6.3 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of semi-continous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```
SemiCont Variable x;
x.lo = 3.2;
x.up = 8.7;

SemiInt Variable y;
y.lo = 5;
y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

Note that Cplex requires a finite upper bound for semi-continous and semi-integer variables.

6.4 Running Out of Memory for MIP problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limited by using the workmem, option with the nodefileind option. Setting nodefileind to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option workmem. That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.

- Set option bttol to a number closer to 1.0 to cause more of a depth-first-search of the tree.
- Set option nodesel to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.
- Set option varsel to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.
- On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using option cuts.

6.5 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as 2^n nodes, where n equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer's memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using some different option settings. Consider some of the shortcuts described previously for improving performance including setting the options for mip gap, objective value difference, upper cutoff, or lower cutoff.

---

7 GAMS/Cplex Log file

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility

value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option simdisplay there is a log line for each refactorization. The screen log has the following appearance:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Using conservative initial basis.

Iteration log . . .
Iteration:     1    Scaled infeas =        193998.067174
Iteration:    29    Objective     =         -3484.286415
Switched to devex.
Iteration:    98    Objective     =         -1852.931117
Iteration:   166    Objective     =          -349.706562

Optimal solution found.

Objective :         901.161538
```

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:     1   Scaled dual infeas =             3.890823
Iteration:    53   Dual objective     =          4844.392441
Iteration:   114   Dual objective     =          1794.360714
Iteration:   176   Dual objective     =          1120.183325
Iteration:   238   Dual objective     =           915.143030
Removing shift (1).

Optimal solution found.

Objective :         901.161538
```

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
```

```
Presolve time =    0.01 sec.
Extracted network with 25 nodes and 116 arcs.
Extraction time =   -0.00 sec.
Iteration log . . .
Iteration:     0   Infeasibility     =          1232.378800 (-1.32326e+12)

Network - Optimal:  Objective =    1.5716820779e+03
Network time =    0.01 sec.  Iterations = 26 (24)

Iteration log . . .
Iteration:     1    Scaled infeas =        212696.154729
Iteration:    62    Scaled infeas =         10020.401232
Iteration:   142    Scaled infeas =          4985.200129
Switched to devex.
Iteration:   217    Objective     =         -3883.782587
Iteration:   291    Objective     =         -1423.126582

Optimal solution found.

Objective :         901.161538
```

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.02 sec.
Number of nonzeros in lower triangle of A*A' = 6545
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.01 sec.
Summary statistics for Cholesky factor:
  Rows in Factor            = 243
  Integer space required    = 578
  Total non-zeros in factor = 8491
  Total FP ops to factor    = 410889
 Itn      Primal Obj        Dual Obj  Prim Inf Upper Inf  Dual Inf
   0  -1.2826603e+06   7.4700787e+08  2.25e+10  6.13e+06  4.00e+05
   1  -2.6426195e+05   6.3552653e+08  4.58e+09  1.25e+06  1.35e+05
   2  -9.9117854e+04   4.1669756e+08  1.66e+09  4.52e+05  3.93e+04
   3  -2.6624468e+04   2.1507018e+08  3.80e+08  1.04e+05  1.20e+04
   4  -1.2104334e+04   7.8532364e+07  9.69e+07  2.65e+04  2.52e+03
   5  -9.5217661e+03   4.2663811e+07  2.81e+07  7.67e+03  9.92e+02
   6  -8.6929410e+03   1.4134077e+07  4.94e+06  1.35e+03  2.16e+02
   7  -8.3726267e+03   3.1619431e+06  3.13e-07  6.84e-12  3.72e+01
   8  -8.2962559e+03   3.3985844e+03  1.43e-08  5.60e-12  3.98e-02
   9  -3.8181279e+03   2.6166059e+03  1.58e-08  9.37e-12  2.50e-02
  10  -5.1366439e+03   2.8102021e+03  3.90e-06  7.34e-12  1.78e-02
  11  -1.9771576e+03   1.5960442e+03  3.43e-06  7.02e-12  3.81e-03
  12  -4.3346261e+02   8.3443795e+02  4.99e-07  1.22e-11  7.93e-04
```

```
  13   1.2882968e+02   5.2138155e+02  2.22e-07  1.45e-11  8.72e-04
  14   5.0418542e+02   5.3676806e+02  1.45e-07  1.26e-11  7.93e-04
  15   2.4951043e+02   6.5911879e+02  1.73e-07  1.43e-11  5.33e-04
  16   2.4666057e+02   7.6179064e+02  7.83e-06  2.17e-11  3.15e-04
  17   4.6820025e+02   8.1319322e+02  4.75e-06  1.78e-11  2.57e-04
  18   5.6081604e+02   7.9608915e+02  3.09e-06  1.98e-11  2.89e-04
  19   6.4517294e+02   7.7729659e+02  1.61e-06  1.27e-11  3.29e-04
  20   7.9603053e+02   7.8584631e+02  5.91e-07  1.91e-11  3.00e-04
  21   8.5871436e+02   8.0198336e+02  1.32e-07  1.46e-11  2.57e-04
  22   8.8146686e+02   8.1244367e+02  1.46e-07  1.84e-11  2.29e-04
  23   8.8327998e+02   8.3544569e+02  1.44e-07  1.96e-11  1.71e-04
  24   8.8595062e+02   8.4926550e+02  1.30e-07  2.85e-11  1.35e-04
  25   8.9780584e+02   8.6318712e+02  1.60e-07  1.08e-11  9.89e-05
  26   8.9940069e+02   8.9108502e+02  1.78e-07  1.07e-11  2.62e-05
  27   8.9979049e+02   8.9138752e+02  5.14e-07  1.88e-11  2.54e-05
  28   8.9979401e+02   8.9139850e+02  5.13e-07  2.18e-11  2.54e-05
  29   9.0067378e+02   8.9385969e+02  2.45e-07  1.46e-11  1.90e-05
  30   9.0112149e+02   8.9746581e+02  2.12e-07  1.71e-11  9.61e-06
  31   9.0113610e+02   8.9837069e+02  2.11e-07  1.31e-11  7.40e-06
  32   9.0113661e+02   8.9982723e+02  1.90e-07  2.12e-11  3.53e-06
  33   9.0115644e+02   9.0088083e+02  2.92e-07  1.27e-11  7.35e-07
  34   9.0116131e+02   9.0116262e+02  3.07e-07  1.81e-11  3.13e-09
  35   9.0116154e+02   9.0116154e+02  4.85e-07  1.69e-11  9.72e-13
Barrier time =    0.39 sec.

Primal crossover.
  Primal:  Fixing 13 variables.
       12 PMoves:  Infeasibility  1.97677059e-06  Objective  9.01161542e+02
        0 PMoves:  Infeasibility  0.00000000e+00  Objective  9.01161540e+02
  Primal:  Pushed 1, exchanged 12.
  Dual:  Fixing 3 variables.
        2 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
        0 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
  Dual:  Pushed 3, exchanged 0.
Using devex.
Total crossover time =    0.02 sec.

Optimal solution found.

Objective :         901.161540
```

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage.

```
Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time =    0.00 sec.

Iteration log . . .
```

```
Iteration:     1   Dual objective     =             0.000000
Root relaxation solution time =    0.01 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0        0.0000    24                      0.0000       40
*     0+    0        6.0000     0        6.0000        0.0000       40  100.00%
*    50+   50        4.0000     0        4.0000        0.0000      691  100.00%
    100    99        2.0000    15        4.0000        0.4000     1448   90.00%
Fixing integer variables, and solving final LP..
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time =    0.00 sec.

Solution satisfies tolerances.

MIP Solution    :            4.000000    (2650 iterations, 185 nodes)
Final LP                :            4.000000    (0 iterations)

Best integer solution possible :           1.000000
Absolute gap                   :                  3
Relative gap                   :                1.5
```

---

8 Detailed Descriptions of Cplex Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the options file is 'cplex.opt'. The options file is case insensitive and the keywords should be given in full.

advind *(integer)*

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS Bratio option can be used to specify when not to use an advanced basis. The Cplex option *advind* can be used to ignore a basis passed on by GAMS (it overrides Bratio). Note that using an advanced basis will disable the Cplex presolve.
(default is determined by GAMS Bratio)

0 do not use advanced basis

1 use advanced basis if available

aggcutlim *(integer)*

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.
Range: non-negative
(default = 3)

aggfill *(integer)*

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the AGGFILL parameter, the substitution will not be made.

(default = 10)

aggind *(integer)*

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.
(default = -1)

-1 once for LP, unlimited for MIP
0 do not use

baralg *(integer)*

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.
(default = 0)

0 Same as 1 for MIP subproblems, 3 otherwise
1 Infeasibility-estimate start
2 Infeasibility-constant start
3 standard barrier algorithm

barcolnz *(integer)*

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns to be considered as dense.
(default = 0)

barcrossalg *(integer)*

Selects which, if any, crossover method is used at the end of a barrier optimization.
(default = 0)

-1 No crossover
0 Automatic
1 Primal crossover
2 Dual crossover

bardisplay *(integer)*

Determines the level of progress information to be displayed while the barrier method is running.
(default = 1)

0 No progress information
1 Display normal information
2 Display diagnostic information

barepcomp *(real)*

Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will terminate

with an optimal solution if the relative complementarity is smaller than this value.
(default = 1.0 e-8)

bargrowth *(real)*

Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem does have an unbounded face.
(default = 1.0 e6)

baritlim *(integer)*

Determines the maximum number of iterations for the barrier algorithm. The default value of -1 allows Cplex to automatically determine the limit based on problem characteristics.
(default = -1)

barmaxcor *(integer)*

Specifies the maximum number of centering corrections that should be done on each iteration. Larger values may improve the numerical performance of the barrier algorithm at the expense of computation time. The default of -1 means the number is automatically determined.
(default = -1)

barobjrng *(real)*

Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.
(default = 1.0 e20)

barooc *(integer)*

Specifies whether or not Cplex should use out-of-core storage (ie disk) for the Barrier Cholesky factorization. This factorization usually accounts for most of the memory usage on large models. Use of this option can therefore extend the range of models that can be solved on a computer with a given amount of RAM. Disk usage with this option is controlled by options workdir and workmem.
(default = 0)

0 off
1 on

barorder *(integer)*

Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of longer ordering runtimes.
(default = 0)

0 Automatic
1 Approximate Minimum Degree (AMD)
2 Approximate Minimum Fill (AMF)
3 Nested Dissection (ND)

barstartalg *(integer)*

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default

starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the predual parameter.
(default = 1)

1 default primal, dual is 0
2 default primal, estimate dual
3 primal average, dual is 0
4 primal average, estimate dual

barthreads *(integer)*

This option sets a limit on the number of threads available to the parallel barrier algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

barvarup *(real)*

Determines an upper bound for all variables that have no finite upper bound. This number is used by the barrier algorithm to detect unbounded optimal faces.
(default = 1.0 e20)

bbinterval *(integer)*

Set interval for selecting a best bound node when doing a best estimate search. Active only when *nodesel* is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting *nodesel* to 1.
(default = 7)

bndstrenind *(integer)*

Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionaly, due to its iterative nature, takes a long time.
(default = -1)

-1 Determine automatically
0 Don't use bound strengthening
1 Use bound strengthening

brdir *(integer)*

Used to decide which branch (up or down) should be taken first at each node.
(default = 0)

-1 Down branch selected first
0 Algorithm decides
1 Up branch selected first

bttol *(real)*

This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the *bttol*

parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used only once a first integer solution is found or when a cutoff has been specified.
Range: [0.0, 1.0]
(default = 0.01)

cliques *(integer)*

Determines whether or not clique cuts should be generated during optimization.
(default = 0)

-1 Do not generate clique cuts
0 Determined automatically
1 Generate clique cuts moderately
2 Generate clique cuts aggressively

coeredind *(integer)*

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of non-integral vertices. However, the linear programs generated at each node may become more difficult to solve.
(default = 2)

0 Do not use coefficient reduction
1 Reduce only to integral coefficients
2 Reduce all potential coefficients

covers *(integer)*

Determines whether or not cover cuts should be generated during optimization.
(default = 0)

-1 Do not generate cover cuts
0 Determined automatically
1 Generate cover cuts moderately
2 Generate cover cuts aggressively

craind *(integer)*

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.
(default = 1)

Primal:

0 ignore objective coefficients during crash
-1, 1 alternate ways of using objective coefficients

Dual:

0, -1 aggressive starting basis
1 default starting basis

cutlo *(real)*

Lower cutoff value for tree search in maximization problems. This is used to cut off any nodes that have an objective value below the lower cutoff value. This option overrides the GAMS Cutoff setting. A too restrictive value may result in no integer solutions being found.
(default = -1 e +75)

cutpass *(integer)*

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.
(default = 0)

-1 None
0 Automatically determined
>0 Maximum passes to perform

cuts *(string)*

Allows generation of all optional cuts to be turned off at once. This is done by changing the meaning of the default value (*0: automatic)* for the various Cplex cut generation options. At the default value of *yes,* cuts whose generation is to be determined automatically are unaffected. If option *cuts* is set to *no,* cuts whose generation is to be determined automatically will not be generated. This allows cuts to be turned off in general while specific cuts are explicitly turned on. The options affected are cliques, covers, disjcuts, flowcovers, flowpaths, fraccuts, gubcovers, implbd, and mircuts.
(default = yes)

cutsfactor *(real)*

This option limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to *cutsfactor* times the original (after presolve) number of rows.
(default = 4.0)

cutup *(real)*

Upper Cutoff value for tree search in minimization problems. This is used to cut off any nodes that have an objective value above the upper cutoff value. This option overrides the GAMS Cutoff setting. A too restrictive value may result in no integer solutions being found.
(default = 1 e +75)

depind *(integer)*

This option determines if the dependency checker will be used.
(default = 0)

0 Do not use the dependency checker
1 Use the dependency checker

disjcuts *(integer)*

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.
(default = 0)

-1 Do not generate disjunctive cuts
0 Determined automatically
1 Generate disjunctive cuts moderately

2 Generate disjunctive cuts aggressively
3 Generate disjunctive cuts very aggressively

dpriind *(integer)*

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual steepest-edge is particularly efficient and does not carry as much computational burden as the primal steepest-edge pricing.
(default = 0)

0 Determined automatically
1 Standard dual pricing
2 Steepest-edge pricing
3 Steepest-edge pricing in slack space
4 Steepest-edge pricing, unit initial norms

epagap *(real)*

Absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epagap setting, the optimization is stopped. This option overrides GAMS OptCA which provides its initial value.
(default = GAMS OptCA)

epgap *(real)*

Relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epgap setting, the mixed integer optimization is stopped. Note the difference in the Cplex definition of the relative tolerance with the GAMS definition. This option overrides GAMS OptCR which provides its initial value.
Range: [0.0, 1.0]
(default = GAMS OptCR)

epint *(real)*

Integrality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.
Range: [1.0e-9, 1.0]
(default = 1.0e-5)

epmrk *(real)*

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.
Range - [0.0001, 0.99999]
(default = 0.01)

epopt *(real)*

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.
Range - [1.0e-9, 1.0e-4]
(default = 1.0e-6)

epper *(integer)*

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the *epper* option to a level where only one perturbation is required. Any value greater than or equal to 1.0e-8 is valid.
(default = 1.0e-6)

eprhs *(real)*

Feasibility tolerance. This specifies the degree to which a problem's basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.
Range - [1.0e-9, 1.0e-4]
(default = 1.0e-6)

flowcovers *(integer)*

Determines whether or not flow cover cuts should be generated during optimization.
(default = 0)

-1 Do not generate flow cover cuts
0 Determined automatically
1 Generate flow cover cuts moderately
2 Generate flow cover cuts aggressively

flowpaths *(integer)*

Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.
(default = 0)

-1 Do not generate flow path cuts
0 Determined automatically
1 Generate flow path cuts moderately
2 Generate flow path cuts aggressively

fraccand *(integer)*

Limits the number of candidate variables for generating Gomory fractional cuts.
(default = 200)

fraccuts *(integer)*

Determines whether or not Gomory fractional cuts should be generated during optimization.
(default = 0)

-1 Do not generate Gomory fractional cuts
0 Determined automatically
1 Generate Gomory fractional cuts moderately
2 Generate Gomory fractional cuts aggressively

fracpass *(integer)*

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter fraccuts is set to a nonzero value.
(default = 0)

0 Automatically determined
>0 Maximum passes to perform

gubcovers *(integer)*

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization. The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.
(default = 0)

-1 Do not generate GUB cover cuts
0 Determined automatically
1 Generate GUB cover cuts moderately
2 Generate GUB cover cuts aggressively

heurfreq *(integer)*

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval.
(default = 0)

-1 Do not use the node heuristic
0 Determined automatically

iis *(string)*

Find an IIS (Irreducably Inconsistent Set of constraints) and write an IIS report to the GAMS solution listing if the model is found to be infeasible. Legal option values are yes or no.
(default = no)

iisind *(integer)*

This option specifies which method to use for finding an IIS (Irreducably Inconsistent Set of constraints). Note that the iis option must be set to cause an IIS computation. This option just specifies the method.
(default = 0)

0 Method with minimum computation time
1 Method generating minimum size for the IIS

implbd *(integer)*

Determines whether or not implied bound cuts should be generated during optimization.
(default = 0)

-1 Do not generate implied bound cuts
0 Determined automatically
1 Generate implied bound cuts moderately
2 Generate implied bound cuts aggressively

interactive *(string)*

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the cplex.opt file. Control is returned to Cplex by entering "continue". The optimization can be aborted by entering "abort". This option can only be used when running from the command line. Legal option values are yes or no.
(default = no)

intsollim *(integer)*

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.
(default = large -- varies by computer)

itlim *(integer)*

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS IterLim option. Any non-negative integer value is valid.
(default = GAMS IterLim)

lpmethod *(integer)*

Specifies which LP algorithm to use. As of Cplex 7.0, the default of Automatic is always Dual Simplex. In future versions, the algorithm may be picked based on problem characteristics.
(default = 0)

0 Automatic
1 Primal Simplex
2 Dual Simplex
3 Network Simplex
4 Barrier

mipdisplay *(integer)*

The amount of information displayed during MIP solution increases with increasing values of this option.
(default = 4)

0 No display
1 Display integer feasible solutions.
2 Displays nodes under mipinterval control.
3 Same as 2 but adds information on cuts.
4 Same as 3 but adds LP display for the root node.
5 Same as 3 but adds LP display for all nodes.

mipemphasis *(integer)*

This option controls whether the tactics for solving a mixed integer programming problem should emphasize feasibility or optimality.
(default = 0)

0 Emphasize finding a proven optimal solution as quickly as possible.
1 Emphasize finding feasible solutions -- likely at the expense of prolonging the time required to find a proven optimal solution.

mipinterval *(integer)*

The MIP interval option value determines the frequency of the node logging when the mipdisplay option is set higher than 1. If the value of the MIP interval option setting is n, then only every nth node, plus all integer feasible nodes, will be logged. This option is useful for selectively limiting the volume of information generated for a problem that requires many nodes to solve. Any non-negative integer value is valid.
(default = 100)

mipordind *(integer)*

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. 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 purchased within the factory. By assigning a higher priority to the build/nobuild decision variable, you can force this logic into the tree search and eliminate wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the *mipordind* option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.

Note: Priorities are assigned to discrete variables using the *.prior* suffix in the GAMS model. Lower *.prior* values mean higher priority. The *.prioropt* model suffix has to be used to signal GAMS to export the priorities to the solver.
(default = 1)

0 Do not use priorities for branching

1 Priority orders are utilized

mipordtype *(integer)*

This option is used to select the type of generic priority order to generate when no priority order is present.
(default = 0)

0 None

1 decreasing cost magnitude

2 increasing bound range

3 increasing cost per coefficient count

mipstart *(integer)*

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization.
(default = 0)

0 do not use the values

1 use the values

mipthreads *(integer)*

This option sets a limit on the number of threads available to the parallel mip algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

mircuts *(integer)*

Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0,

generation is continued only if it seems to be helping.
(default = 0)

-1 Do not generate MIR cuts
0 Determined automatically
1 Generate MIR cuts moderately
2 Generate MIR cuts aggressively

names *(string)*

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.
(default = yes)

no Do not load GAMS names into Cplex
yes Load GAMS names into Cplex

netdisplay *(integer)*

This option controls the log for network iterations.
(default = 2)

0 No network log.
1 Displays true objective values.
2 Displays penalized objective values.

netepopt *(real)*

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.
Range - [1.0e-11, 1.0e-4]
(default = 1.0e-6)

neteprhs *(real)*

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.
Range - [1.0e-11, 1.0e-4]
(default = 1.0e-6)

netfind *(integer)*

Specifies the level of network extraction to be done.
(default = 2)

1 Extract pure network only
2 Try reflection scaling
3 Try general scaling

netitlim *(integer)*

Iteration limit for the network simplex method.
(default = large -- varies by computer)

netppriind *(integer)*

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.
(default = 0)

0 Automatic
1 Partial pricing
2 Multiple partial pricing
3 Multiple partial pricing with sorting

nodefileind *(integer)*

Specifies how node files are handled during MIP processing. Used when parameter workmem has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node "files" in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter workdir and Cplex actively manages which nodes remain in memory for processing.
(default = 1)

0 No node files
1 Node files in memory and compressed
2 Node files on disk
3 Node files on disk and compressed

nodelim *(integer)*

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix.

nodesel *(integer)*

This option is used to set the rule for selecting the next node to process when backtracking.
(default = 1)

0 Depth-first search. This chooses the most recently created node.
1 Best-bound search. This chooses the unprocessed node with the best objective function for the associated LP relaxation.
2 Best-estimate search. This chooses the node with the best estimate of the integer objective value that would be obtained once all integer infeasibilities are removed.
3 Alternate best-estimate search.

objdif *(real)*

A means for automatically updating the cutoff to more restrictive values. Normally the most recently found integer feasible solution objective value is used as the cutoff for subsequent nodes. When this option is set to a positive value, the value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. The option can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions. This option overrides the GAMS Cheat parameter.
(default = 0.0)

objllim *(real)*

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.
(default = -1.0e+75)

objrng *(string)*

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, **objrng** can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying "all" will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.
(default = no objective ranging is done)

objulim *(real)*

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.
(default = 1.0e+75)

perind *(integer)*

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting perind to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.
(default = 0)

0 not automatically perturbed

1 automatically perturbed

perlim *(integer)*

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.
(default = 0)

ppriind *(integer)*

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use *dpriind* to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.
(default = 0)

-1 Reduced-cost pricing. This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column).

0 Hybrid reduced-cost and Devex pricing.

1 Devex pricing. This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous.

2 Steepest edge pricing. If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems.

3 Steepest edge pricing with slack initial norms. This reduces the computationally intensive nature of steepest edge pricing.

4 Full pricing

precompress *(integer)*

Compress the original model after presolve is performed. This can save a considerable amount of memory for large models. Under the default setting, Cplex will decide whether or not to perform the compression based on model characteristics.
(default = 0)

-1 off

0 automatic

1 on

predual *(integer)*

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The *predual* option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.
(default = 0)

-1 do not give dual to optimizer

0 automatic

1 give dual to optimizer

preind *(integer)*

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, if may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.
(default = 1)

0 Do not presolve the problem

1 Perform the presolve

prepass *(integer)*

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set preind to 0 to turn both off.
(default = -1)

-1 Determined automatically

0 No presolve

preslvnd *(integer)*

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.
(default = 0)

-1 No node presolve
0 Automatic
1 Force node presolve

pricelim *(integer)*

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative integer values are valid.
(default = 0, in which case it is determined automatically)

printoptions *(string)*

Write the values of all options to the GAMS listing file. Valid values are no or yes.
(default = no)

probe *(integer)*

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.
(default = 0)

-1 no probing
0 automatic
1 limited probing
2 more probing
3 full probing

quality *(string)*

Write solution quality statistics to the listing file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing.
(default = no)

reduce *(integer)*

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.
(default = 3)

0 No primal or dual reductions
1 Only primal reductions
2 Only dual reductions
3 Both primal and dual reductions

reinv *(integer)*

Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex's performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.
(default = 0, in which case it is determined automatically)

relaxpreind *(integer)*

This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.
(default = 0)

0 do not presolve initial relaxation

1 use presolve on initial relaxation

relobjdif *(real)*

The relative version of the objdif option. Ignored if objdif is non-zero.
(default = 0)

rerun *(string)*

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with presolve turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The iis option can be used to get even more diagnostic information. The *rerun* option controls this behavior. Valid values are auto, yes or no. The default value of auto is equivalent to no if names are successfully loaded into Cplex. In that case the Cplex messages from presolve identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to yes. Loading of GAMS names into Cplex is controlled by option names.
(default = auto)

rhsrng *(string)*

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, **rhsrng** can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS equation named. Specifying "all" will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.
(default = no right-hand-side ranging is done)

rngrestart *(string)*

Write ranging information, in GAMS readable format, to the file named. Options objrng and rhsrng are used to specify which GAMS variables or equations are included.
(default = ranging information is printed to the listing file)

scaind *(integer)*

This option influences the scaling of the problem matrix.
(default = 0)

-1 No scaling

0 Standard scaling - An equilibration scaling method is implemented which is generally very effective.

1 Modified, more aggressive scaling method that can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.

simdisplay *(integer)*

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.
(default = 1)

0 No iteration messages are issued until the optimal solution is reported.
1 An iteration log message will be issued after each refactorization. Each entry will contain the iteration count and scaled infeasibility or objective value.
2 An iteration log message will be issued after each iteration. The variables, slacks and artificials entering and leaving the basis will also be reported.

simthreads *(integer)*

This option sets a limit on the number of threads available to the parallel simplex algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

singlim *(integer)*

The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that has been found. Any non-negative integer value is valid.
(default = 10)

startalg *(integer)*

Selects the algorithm to use for the initial relaxation of a MIP.
(default = 2)

1 primal simplex
2 dual simplex
3 network followed by dual simplex
4 barrier with crossover
5 dual simplex to iteration limit, then barrier
6 barrier without crossover

strongcandlim *(integer)*

Limit on the length of the candidate list for strong branching (varsel = 3).
(default = 10)

strongitlim *(integer)*

Limit on the number of iterations per branch in strong branching (varsel = 3). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.
(default = 0)

strongthreadlim *(integer)*

Controls the number of parallel threads available for strong branching (varsel = 3). This option does nothing if option mipthreads is greater than 1.
(default = 1)

subalg *(integer)*

Strategy for solving linear sub-problems at each node.

(default = 2)

1 primal simplex
2 dual simplex
3 network optimizer followed by dual simplex
4 barrier with crossover
5 dual simplex to iteration limit, then barrier
6 barrier without crossover

tilim *(real)*

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.
(default = GAMS ResLim)

trelim *(real)*

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.
(default = 1.0e+75)

varsel *(integer)*

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.
(default = 0)

-1 Branch on variable with minimum infeasibility. This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.
0 Branch variable automatically selected.
1 Branch on variable with maximum infeasibility. This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.
2 Branch based on pseudo costs. Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.
3 Strong Branching. This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.
4 Branch based on pseudo reduced costs.

workdir *(character string)*

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

workmem *(real)*

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter workdir.

writebas *(character string)*

Write a basis file.

writelp *(character string)*

Write a file in Cplex LP format.

writemps *(character string)*

Write an MPS problem file.

writeord *(character string)*

Write a Cplex ord (containing priority and branch direction information) file.

writesav *(character string)*

Write a binary problem file.

writesos *(character string)*

Write a file containing the SOS structure. For models with SOS variables only.

Last modified November 16, 2001 by PES

GAMS/DECIS*USER'S GUIDE

Gerd Infanger†

1999

Abstract

This User's Guide discusses how to run DECIS directly from GAMS. It describes how to set up stochastic problems using the GAMS modeling language and DECIS as the optimizer for their solution. DECIS is a system for solving large-scale stochastic programs. It includes a variety of solution strategies and can solve problems with numerous stochastic parameters. DECIS interfaces either with MINOS or CPLEX for solving subproblems. GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS makes the formulation of stochastic problems easy and gives you all the power and flexibility of using an integrated modeling system solver environment.

†Dr. Gerd Infanger is an Associate Professor at Vienna University of Technology and a Consulting Professor at Stanford University.

Contents

1. DECIS

1.1. Introduction

DECIS is a system for solving large-scale stochastic programs, programs, which include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to be known by their probability distribution. It employs Benders decomposition and allows using advanced 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 pre-sampling. When using Monte Carlo sampling the user has the option of employing crude Monte Carlo without variance reduction techniques, or using as variance reduction techniques importance sampling or control variates, based on either an additive or a multiplicative approximation function. Pre-sampling is limited to using crude Monte Carlo only.

For solving linear and nonlinear programs (master and subproblems arising from the decomposition) DECIS interfaces with MINOS or CPLEX. MINOS, see Murtagh and Saunders (1983) [5], is a state-of-the-art solver for large-scale linear and nonlinear programs, and CPLEX, see CPLEX Optimization, Inc. (1989–1997) [2], is one of the fastest linear programming solvers available.

For details about the DECIS system consult the DECIS User's Guide, see Infanger (1997) [4]. It includes a comprehensive mathematical description of the methods used by DECIS. In this Guide we concentrate on how to use DECIS directly from GAMS, see Brooke, A., Kendrik, D. and Meeraus, A. (1988) [1], and especially on how to model stochastic programs using the GAMS/DECIS interface. First, however, in section 1.2 we give a brief description of what DECIS can do and what solution strategies it uses. This description has been adapted from the DECIS User's Guide. In section 2 we discuss in detail how to set up a stochastic problem using GAMS/DECIS and give a description of the parameter setting and outputs obtained. In Appendix A we show the GAMS/DECIS formulation of two illustrative examples (APL1P and APL1PC) discussed in the DECIS User's Guide. A list of DECIS error messages are represented in Appendix B.

1.2. What DECIS can do

DECIS solves two-stage stochastic linear programs with recourse:

$$\begin{array}{rrrrrrrl} \min\ z & = & cx & + & E\ f^\omega y^\omega & & & \\ s/t & & Ax & & & = & b & \\ & & -B^\omega x & + & D^\omega y^\omega & = & d^\omega & \\ & & x, & & y^\omega & \geq & 0, & \omega \in \Omega. \end{array}$$

where x denotes the first-stage, y^ω the second-stage decision variables, c represents the first-stage and f^ω the second-stage objective coefficients, A, b represent the coefficients and right hand sides of the first-stage constraints, and B^ω, D^ω, d^ω represent the parameters of the second-stage constraints, where the transition matrix B^ω couples the two stages. In the literature D^ω is often referred to as the technology matrix or recourse matrix. The first stage parameters are known with certainty. The second stage parameters are random parameters that assume outcomes labeled ω with probability $p(\omega)$, where Ω denotes the set of all possible outcome labels.

At the time the first-stage decision x has to be made, the second-stage parameters are only known by their probability distribution of possible outcomes. Later after x is already determined, an actual outcome of the second-stage parameters will become known, and the second-stage decision y^ω is made based on knowledge of the actual outcome ω. The objective is to find a feasible decision x that minimizes the total expected costs, the sum of first-stage costs and expected second-stage costs.

For discrete distributions of the random parameters, the stochastic linear program can be represented by the corresponding *equivalent deterministic linear program*:

$$\begin{array}{rrrrrrrrrrrrr}
\min\ z & = & cx & + & p^1fy^1 & + & p^2fy^2 & + & \cdots & + & p^Wfy^W & & \\
s/t & & Ax & & & & & & & & & = & b \\
 & & -B^1x & + & Dy^1 & & & & & & & = & d^1 \\
 & & -B^2x & & & + & Dy^2 & & & & & = & d^2 \\
 & & \vdots & & & & & & \ddots & & & & \vdots \\
 & & -B^Wx & & & & & & & + & Dy^W & = & d^W \\
 & & x, & & y^1, & & y^2, & & \ldots, & & y^W & \geq & 0,
\end{array}$$

which contains all possible outcomes $\omega \in \Omega$. Note that for practical problems W is very large, e.g., a typical number could be 10^{20}, and the resulting equivalent deterministic linear problem is too large to be solved directly.

In order to see the two-stage nature of the underlying decision making process the folowing representation is also often used:

$$\begin{array}{rrrrr}
\min\ cx & + & E\ z^\omega(x) & & \\
Ax & & & = & b \\
x & & & \geq & 0
\end{array}$$

where

$$\begin{array}{rrrrr}
z^\omega(x) & = & \min\ f^\omega y^\omega & & \\
 & & D^\omega y^\omega & = & d^\omega + B^\omega x \\
 & & y^\omega & \geq & 0,\ \omega \in \Omega = \{1, 2, \ldots, W\}.
\end{array}$$

DECIS employs different strategies to solve two-stage stochastic linear programs. It computes an exact optimal solution to the problem or approximates the true optimal solution very closely and gives a confidence interval within which the true optimal objective lies with, say, 95% confidence.

1.3. Representing uncertainty

It is favorable to represent the uncertain second-stage parameters in a structure. Using $V = (V_1, \ldots, V_h)$ an h-dimensional independent random vector parameter that assumes outcomes $v^\omega = (v_1, \ldots, v_h)^\omega$ with probability $p^\omega = p(v^\omega)$, we represent the uncertain second-stage parameters of the problem as functions of the independent random parameter V:

$$f^\omega = f(v^\omega), \quad B^\omega = B(v^\omega), \quad D^\omega = D(v^\omega), \quad d^\omega = d(v^\omega).$$

Each component V_i has outcomes $v_i^{\omega_i}$, $\omega_i \in \Omega_i$, where ω_i labels a possible outcome of component i, and Ω_i represents the set of all possible outcomes of component i. An outcome

of the random vector

$$v^{\omega} = (v_1^{\omega_1}, \ldots, v_h^{\omega_h})$$

consists of h independent component outcomes. The set

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_h$$

represents the crossing of sets Ω_i. Assuming each set Ω_i contains W_i possible outcomes, $|\Omega_i| = W_i$, the set Ω contains $W = \prod W_i$ elements, where $|\Omega| = W$ represents the number of all possible outcomes of the random vector V. Based on independence, the joint probability is the product

$$p^{\omega} = p_1^{\omega_1} p_2^{\omega_2} \cdots p_h^{\omega_h}.$$

Let η denote the vector of all second-stage random parameters, e.g., $\eta = \text{vec}(f, B, D, d)$. The outcomes of η may be represented by the following general linear dependency model:

$$\eta^{\omega} = \text{vec}(f^{\omega}, B^{\omega}, d^{\omega}, d^{\omega}) = Hv^{\omega}, \quad \omega \in \Omega$$

where H is a matrix of suitable dimensions. DECIS can solve problems with such general linear dependency models.

1.4. Solving the universe problem

We refer to the universe problem if we consider all possible outcomes $\omega \in \Omega$ and solve the corresponding problem exactly. This is not always possible, because there may be too many possible realizations $\omega \in \Omega$. For solving the problem DECIS employs Benders decomposition, splitting the problem into a master problem, corresponding to the first-stage decision, and into subproblems, one for each $\omega \in \Omega$, corresponding to the second-stage decision. The details of the algorithm and techniques used for solving the universe problem are discussed in The DECIS User's Manual.

Solving the universe problem is referred to as strategy 4. Use this strategy only if the number of universe scenarios is reasonably small. There is a maximum number of universe scenarios DECIS can handle, which depends on your particular resources.

1.5. Solving the expected value problem

The expected value problem results from replacing the stochastic parameters by their expectation. It is a linear program that can also easily be solved by employing a solver directly. Solving the expected value problem may be useful by itself (for example as a benchmark to compare the solution obtained from solving the stochastic problem), and it also may yield a good starting solution for solving the stochastic problem. DECIS solves the expected value problem using Benders decomposition. The details of generating the expected value problem and the algorithm used for solving it are discussed in the DECIS User's Manual. To solve the expected value problem choose strategy 1.

1.6. Using Monte Carlo sampling

As noted above, for many practical problems it is impossible to obtain the universe solution, because the number of possible realizations $|\Omega|$ is way too large. The power of DECIS lies in its ability to compute excellent approximate solutions by employing Monte Carlo sampling techniques. Instead of computing the expected cost and the coefficients and the right-hand sides of the Benders cuts exactly (as it is done when solving the universe problem), DECIS, when using Monte Carlo sampling, estimates the quantities in each iteration using an independent sample drawn from the distribution of the random parameters. In addition to using crude Monte Carlo, DECIS uses importance sampling or control variates as variance reduction techniques.

The details of the algorithm and the different techniques used are described in the DECIS User's Maual. You can choose crude Monte Carlo, referred to as strategy 6, Monte Carlo importance sampling, referred to as strategy 2, or control variates, referred to as strategy 10. Both Monte Carlo importance sampling and control variates have been shown for many problems to give a better approximation compared to employing crude Monte Carlo sampling.

When using Monte Carlo sampling DECIS computes a close approximation to the true solution of the problem, and estimates a close approximation of the true optimal objective value. It also computes a confidence interval within which the true optimal objective of the problem lies, say with 95% confidence. The confidence interval is based on rigorous statistical theory. An outline of how the confidence interval is computed is given in the DECIS User's Manual. The size of the confidence interval depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for the estimation. You can expect the confidence interval to be very small, especially when you employ importance sampling or control variates as a variance reduction technique.

When employing Monte Carlo sampling techniques you have to choose a sample size (set in the parameter file). Clearly, the larger the sample size the better will be the approximate solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Setting the sample size too small may lead to bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

1.7. Monte Carlo pre-sampling

We refer to pre-sampling when we first take a random sample from the distribution of the random parameters and then generate the approximate stochastic problem defined by the sample. The obtained approximate problem is then solved exactly using decomposition. This is in contrast to the way we used Monte Carlo sampling in the previous section, where we used Monte Carlo sampling in each iteration of the decomposition.

The details of the techniques used for pre-sampling are discussed in the DECIS User's Manual. DECIS computes the exact solution of the sampled problem using decomposition. This solution is an approximate solution of the original stochastic problem. Besides this approximate solution, DECIS computes an estimate of the expected cost corresponding to this approximate solution and a confidence interval within which the true optimal objective of the original stochastic problem lies with, say, 95% confidence. The confidence interval is

based on statistical theory, its size depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for generating the approximate problem. In conjunction with pre-sampling no variance reduction techniques are currently implemented.

Using Monte Carlo pre-sampling you have to choose a sample size. Clearly, the larger the sample size you choose, the better will be the solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Again, setting the sample size as too small may lead to a bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

For using Monte Carlo pre-sampling choose strategy 8.

1.8. Regularized decomposition

When solving practical problems, the number of Benders iterations can be quite large. In order to control the decomposition, with the hope to reduce the iteration count and the solution time, DECIS makes use of regularization. When employing regularization, an additional quadratic term is added to the objective of the master problem, representing the square of the distance between the best solution found so far (the incumbent solution) and the variable x. Using this term, DECIS controls the distance of solutions in different decomposition iterations.

For enabling regularization you have to set the corresponding parameter. You also have to choose the value of the constant rho in the regularization term. The default is regularization disabled. Details of how DECIS carries out regularization are represented in the DECIS User's Manual.

Regularization is only implemented when using MINOS as the optimizer for solving subproblems. Regularization has proven to be helpful for problems that need a large number of Benders iteration when solved without regularization. Problems that need only a small number of Benders iterations without regularization are not expected to improve much with regularization, and may need even more iterations with regularization than without.

2. GAMS/DECIS

GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS spares you from worrying about all the details of the input formats. It makes the problem formulation much easier but still gives you almost all the flexibility of using DECIS directly.

The link from GAMS to DECIS has been designed in such a way that almost no extensions to the GAMS modeling language were necessary for carrying out the formulation and solution of stochastic programs. In a next release of GAMS, however, additions to the language are planned that will allow you to model stochastic programs in an even more elegant way.

2.1. Setting up a stochastic program using GAMS/DECIS

The interface from GAMS to DECIS supports the formulation and solution of stochastic *linear* programs. DECIS solves them using two-stage decomposition. The GAMS/DECIS

interface resembles closely the structure of the SMPS (stochastic mathematical programming interface) discussed in the DECIS User's Manual. The specification of a stochastic problem using GAMS/DECIS uses the following components:

- the deterministic (core) model,
- the specification of the decision stages,
- the specification of the random parameters, and
- setting DECIS to be the optimizer to be used.

2.2. Starting with the deterministic model

The core model is the deterministic linear program where all random parameters are replaced by their mean or by a particular realization. One could also see it as a GAMS model model without any randomness. It could be a deterministic model that you have, which you intend to expand to a stochastic one. Using DECIS with GAMS allows you to easily extend a deterministic linear programming model to a stochastic one. For example, the following GAMS model represents the a deterministic version of the electric power expansion planning illustrative example discussed in Infanger (1994).

```
*   APL1P test model
*   Dr. Gerd Infanger, November 1997
*   Deterministic Program

set g generators / g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
             h    m    l
   g1        4.3  2.0  0.5
   g2        8.7  4.0  1.0;

parameter d(dl) demand                   / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost           total cost;
positive variable x(g)        capacity of generators;
positive variable y(g, dl)    operating level;
positive variable s(dl)       unserved demand;

equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;
```

```
cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

option lp=minos5;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

2.3. Setting the decision stages

Next in order to extend a deterministic model to a stochastic one you must specify the decision stages. DECIS solves stochastic programs by two-stage decomposition. Accordingly, you must specify which variables belong to the first stage and which to the second stage, as well as which constraints are first-stage constraints and which are second-stage constraints. First stage constraints involve only first-stage variables, second-stage constraints involve both first- and second-stage variables. You must specify the stage of a variable or a constraint by setting the stage suffix ".STAGE" to either one or two depending on if it is a first or second stage variable or constraint. For example, expanding the illustrative model above by

```
* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;
```

would make x(g) first-stage variables, y(g, dl) and s(dl) second-stage variables, cmin(g) and cmax(g) first-stage constraints, and omax(g) and demand(g) second-stage constraints. The objective is treated separately, you don't need to set the stage suffix for the objective variable and objective equation.

It is noted that the use of the `.stage` variable and equation suffix causes the GAMS scaling facility through the `.scale` suffices to be unavailable. Stochastic models have to be scaled manually.

2.4. Specifying the stochastic model

DECIS supports any linear dependency model, i.e., the outcomes of an uncertain parameter in the linear program are a linear function of a number of independent random parameter outcomes. DECIS considers only discrete distributions, you must approximate any continuous distributions by discrete ones. The number of possible realizations of the discrete random parameters determines the accuracy of the approximation. A special case of a linear dependency model arises when you have only independent random parameters in your model. In this case the independent random parameters are mapped one to one into the random parameters of the stochastic program. We will present the independent case first and then expand to the case with linear dependency. According to setting up a linear dependency model we present the formulation in GAMS by first defining independent random parameters and then defining the distributions of the uncertain parameters in your model.

2.4.1. Specifying independent random parameters

There are of course many different ways you can set up independent random parameters in GAMS. In the following we show one possible way that is generic and thus can be adapted for different models. The set-up uses the set stoch for labeling outcome named "out" and probability named "pro" of each independent random parameter. In the following we show how to define an independent random parameter, say, v1. The formulation uses the set omega1 as driving set, where the set contains one element for each possible realization the random parameter can assume. For example, the set omega1 has four elements according to a discrete distribution of four possible outcomes. The distribution of the random parameter is defined as the parameter v1, a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible realizations of the set omega1, "o11", "o12", "o13", and "o14". For example, the random parameter v1 has outcomes of $-1.0, -0.9, -0.5, -0.1$ with probabilities $0.2, 0.3, 0.4, 0.1$, respectively. Instead of using assignment statements for inputting the different realizations and corresponding probabilities you could also use the table statement. You could also the table statement would work as well. Always make sure that the sum of the probabilities of each independent random parameter adds to one.

```
* defining independent stochastic parameters
set stoch /out, pro /;
set omega1 / o11, o12, o13, o14 /;

table v1(stoch, omega1)
      o11   o12   o13   o14
out  -1.0  -0.9  -0.5  -0.1
pro   0.2   0.3   0.4   0.1
;
```

Random parameter v1 is the first out of five independent random parameters of the illustrative model APL1P, where the first two represent the independent availabilities of the generators g1 and g2 and the latter three represent the independent demands of the demand levels h, m, and l. We also represent the definitions of the remaining four independent random parameters. Note that random parameters v3, v4, and v5 are identically distributed.

```
set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21   o22   o23   o24   o25
out  -1.0  -0.9  -0.7  -0.1  -0.0
pro   0.1   0.2   0.5   0.1   0.1
;


set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;


set omega4 / o41, o42, o43, o44 /;
table v4(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;
```

2.4.2. Defining the distributions of the uncertain parameters in the model

Having defined the independent stochastic parameters (you may copy the setup above and adapt it for your model), we next define the stochastic parameters in the GAMS model. The stochastic parameters of the model are defined by writing a file, the GAMS stochastic file, using the put facility of GAMS. The GAMS stochastic file resembles closely the stochastic file of the SMPS input format. The main difference is that we use the row, column, bounds, and right hand side names of the GAMS model and that we can write it in free format.

Independent stochastic parameters

First we describe the case where all stochastic parameters in the model are independent, see below the representation of the stochastic parameters for the illustrative example APL1P, which has five independent stochastic parameters.

First define the GAMS stochastic file "MODEL.STG" (only the exact name in uppercase letters is supported) and set up GAMS to write to it. This is done by the first two statements. You may want to consult the GAMS manual for how to use put for writing files. The next statement "INDEP DISCRETE" indicates that a section of independent stochastic parameters follows. Then we write all possible outcomes and corresponding probabilities for each stochastic parameter best by using a loop statement. Of course one

could also write each line separately, but this would not look nicely. Writing a "*" between the definitions of the independent stochastic parameters is merely for optical reasons and can be omitted.

```
* defining distributions (writing file MODEL.STG)
file stg /MODEL.STG/;
put stg;

put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1   ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
);
put "*" /;
loop(omega2,
put "x g2  omax g2   ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS  demand h   ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
);
put "*" /;
loop(omega4,
put "RHS  demand m   ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
)
put "*" /;
loop(omega5,
put "RHS  demand l   ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
);
putclose stg;
```

In the example APL1P the first stochastic parameter is the availability of generator g1. In the model the parameter appears as the coefficient of variable x(g1) in equation omax(g1). The definition using the put statement first gives the stochastic parameter as the intersection of variable x(g1) with equation omax(g1), but without having to type the braces, thus *x g1 omax g1*, then the outcome *v1("out", omega1)* and the probability *v1("pro", omega1)* separated by *"period2"*. The different elements of the statement must be separated by blanks. Since the outcomes and probabilities of the first stochastic parameters are driven by the set omega1 we loop over all elements of the set omega1. We continue and define all possible outcomes for each of the five independent stochastic parameters.

In the example of independent stochastic parameters, the specification of the distribution of the stochasic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
INDEP DISCRETE
x g1  omax g1          -1.00 period2         0.20
x g1  omax g1          -0.90 period2         0.30
x g1  omax g1          -0.50 period2         0.40
x g1  omax g1          -0.10 period2         0.10
*
x g2  omax g2          -1.00 period2         0.10
x g2  omax g2          -0.90 period2         0.20
x g2  omax g2          -0.70 period2         0.50
```

```
x g2  omax g2          -0.10 period2         0.10
x g2  omax g2           0.00 period2         0.10
*
RHS  demand h         900.00 period2         0.15
RHS  demand h        1000.00 period2         0.45
RHS  demand h        1100.00 period2         0.25
RHS  demand h        1200.00 period2         0.15
*
RHS  demand m         900.00 period2         0.15
RHS  demand m        1000.00 period2         0.45
RHS  demand m        1100.00 period2         0.25
RHS  demand m        1200.00 period2         0.15
*
RHS  demand l         900.00 period2         0.15
RHS  demand l        1000.00 period2         0.45
RHS  demand l        1100.00 period2         0.25
RHS  demand l        1200.00 period2         0.15
```

For defining stochastic parameters in the right-hand side of the model use the keyword *RHS* as the column name, and the equation name of the equation which right-hand side is uncertain, see for example the specification of the uncertain demands *RHS demand h*, *RHS demand m*, and *RHS demand l*. For defining uncertain bound parameters you would use the keywords *UP*, *LO*, or *FX*, the string *bnd*, and the variable name of the variable, which upper, lower, or fixed bound is uncertain.

Note all the keywords for the definitions are in capital letters, i.e., "INDEP DISCRETE", "RHS", and not represented in the example "UP", "LO", and "FX".

It is noted that in GAMS equations, variables may appear in the right-hand side, e.g. "`EQ..  X+1 =L= 2*Y`". When the coefficient 2 is a random variable, we need to be aware that GAMS will generate the following LP row `X - 2*Y =L= -1`. Suppose the probability distribution of this random variable is given by:

```
set s scenario /pessimistic, average, optimistic/;
parameter outcome(s) / pessimistic 1.5
                       average     2.0
                       optimistic  2.3 /;
parameter prob(s)    / pessimistic 0.2
                       average     0.6
                       optimistic  0.2 /;
```

then the correct way of generating the entries in the stochastic file would be:

```
loop(s,
   put  "Y EQ ",(-outcome(s))," PERIOD2 ",prob(s)/;
);
```

Note the negation of the outcome parameter. Also note that expressions in a PUT statement have to be surrounded by parentheses. GAMS reports in the *row listing* section of the listing file how equations are generated. You are encouraged to inspect the row listing how coefficients appear in a generated LP row.

Dependent stochastic parameters

Next we describe the case of general linear dependency of the stochastic parameters in the model, see below the representation of the stochastic parameters for the illustrative example APL1PCA, which has three dependent stochastic demands driven by two independent stochastic random parameters. First we give the definition of the two independent stochastic parameters, which in the example happen to have two outcomes each.

```
* defining independent stochastic parameters
set stoch /out, pro/;

set omega1 / o11, o12 /;
table v1(stoch,omega1)
       o11  o12
out    2.1  1.0
pro    0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
       o21  o22
out    2.0  1.0
pro    0.2  0.8 ;
```

We next define the parameters of the transition matrix from the independent stochastic parameters to the dependent stochastic parameters of the model. We do this by defining two parameter vectors, where the vector $hm1$ gives the coefficients of the independent random parameter $v1$ in each of the three demand levels and the vector $hm2$ gives the coefficients of the independent random parameter $v2$ in each of the three demand levels.

```
parameter hm1(dl) / h  300., m  400., l  200. /;
parameter hm2(dl) / h  100., m  150., l  300. /;
```

Again first define the GAMS stochastic file "MODEL.STG" and set GAMS to write to it. The statement *BLOCKS DISCRETE* indicates that a section of linear dependent stochastic parameters follows.

```
* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL  v1   period2 ", v1("pro", omega1)/;
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2   period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
```

```
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;
```

Dependent stochastic parameters are defined as functions of independent random parameters. The keyword *BL* labels a possible realization of an independent random parameter. The name besides the *BL* keyword is used to distinguish between different outcomes of the same independent random parameter or a different one. While you could use any unique names for the independent random parameters, it appears natural to use the names you have already defined above, e.g., $v1$ and $v2$. For each realization of each independent random parameter define the outcome of every dependent random parameter (as a function of the independent one). If a dependent random parameter in the GAMS model depends on two or more different independent random parameter the contributions of each of the independent parameters are added. We are therefore in the position to model any linear dependency model. (Note that the class of models that can be accommodated here is more general than linear. The functions, with which an independent random variable contributes to the dependent random variables can be any ones in one argument. As a general rule, any stochastic model that can be estimated by linear regression is supported by GAMS/DECIS.)

Define each independent random parameter outcome and the probability associated with it. For example, the statement starting with *BL v1 period2* indicates that an outcome of (independent random parameter) *v1* is being defined. The name *period2* indicates that it is a second-stage random parameter, and *v1("pro", omega1)* gives the probability associated with this outcome. Next list all random parameters dependent on the independent random parameter outcome just defined. Define the dependent stochastic parameter coefficients by the GAMS variable name and equation name, or "RHS" and variable name, together with the value of the parameter associated with this realization. In the example, we have three dependent demands. Using the scalar $h1$ for intermediately storing the results of the calculation, looping over the different demand levels *dl* we calculate *h1 = hm1(dl) * v1("out", omega1)* and define the dependent random parameters as the right-hand sides of equation $demand(dl)$.

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when *BL v1* appears the second time), a different outcome of the same independent random parameter is being defined, while a different block name (e.g., when *BL v2* appears the first time) indicates that the first outcome of a different independent random parameter is being defined. You must ensure that the probabilities of the different outcomes of each of the independent random parameters add up to one. The loop over all elements of $omega1$ defines all realizations of the independent random parameter $v1$ and the loop over all elements of $omega2$ defines all realizations of the independent random parameter $v2$.

Note for the first realization of an independent random parameter, you *must* define all dependent parameters and their realizations. The values entered serve as a base case. For any other realization of an independent random parameter you only need to define the dependent parameters that have different coefficients than have been defined in the base case. For those not defined in a particular realization, their values of the base case are automatically added.

In the example of dependent stochastic parameters above, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
BLOCKS DISCRETE
BL  v1   period2        0.50
RHS demand h       630.00
RHS demand m       840.00
RHS demand l       420.00
BL  v1   period2        0.50
RHS demand h       300.00
RHS demand m       400.00
RHS demand l       200.00
 BL v2   period2        0.20
RHS demand h       200.00
RHS demand m       300.00
RHS demand l       600.00
 BL v2   period2        0.80
RHS demand h       100.00
RHS demand m       150.00
RHS demand l       300.00
```

Again all the keywords for the definitions are in capital letters, i.e., "BLOCKS DISCRETE", "BL", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that you can only define random parameter coefficients that are nonzero in your GAMS model. When setting up the deterministic core model put a nonzero entry as a placeholder for any coefficient that you wish to specify as a stochastic parameter. Specifying a random parameter at the location of a zero coefficient in the GAMS model causes DECIS to terminate with an error message.

2.5. Setting DECIS as the optimizer

After having finished the stochastic definitions you must set DECIS as the optimizer. This is done by issuing the following statements:

```
* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;
```

The statement *option lp = decis* sets DECIS as the optimizer to be used for solving the stochastic problem. Note that if you do not use DECIS, but instead use any other linear programming optimizer, your GAMS model will still run and optimize the deterministic core model that you have specified. The statement *apl1p.optfile = 1* forces GAMS to process the file DECIS.OPT, in which you may define any DECIS parameters.

2.5.1. Setting parameter options in the GAMS model

The options iteration limit and resource limit can be set directly in your GAMS model file. For example, the following statements

```
option iterim = 1000;
option reslim = 6000;
```

constrain the number of decomposition iterations to be less than or equal to 1000, and the elapsed time for running DECIS to be less than or equal to 6000 seconds or 100 minutes.

2.5.2. Setting parameters in the DECIS options file

In the DECIS options file DECIS.OPT you can specify parameters regarding the solution algorithm used and control the output of the DECIS program. There is a record for each parameter you want to specify. Each record consists of the value of the parameter you want to specify and the keyword identifying the parameter, separated by a blank character or a comma. You may specify parameters with the following keywords: "istrat", "nsamples", "nzrows", "iwrite", "ibug", "iscratch", "ireg", "rho", "tolben", and "tolw" *in any order*. Each keyword can be specified in lower case or upper case text in the format (A10). Since DECIS reads the records in free format you don't have to worry about the format, but some computers require that the text is inputted in quotes. Parameters that are not specified in the parameter file automatically assume their default values.

istrat — Defines the solution strategy used. The default value is istrat = 3.

- istrat = 1 Solves the expected value problem. All stochastic parameters are replaced by their expected values and the corresponding deterministic problem is solved using decomposition.
- istrat = 2 Solves the stochastic problem using Monte Carlo importance sampling. You have to additionally specify what approximation function you wish to use, and the sample size used for the estimation, see below.
- istrat = 3 Refers to istrat = 1 plus istrat = 2. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using importance sampling.
- istrat = 4 Solves the stochastic universe problem by enumerating all possible combinations of realizations of the second-stage random parameters. It gives you the exact solution of the stochastic program. This strategy may be impossible, because there may be way too many possible realizations of the random parameters.
- istrat = 5 Refers to istrat = 1 plus istrat = 4. First solves the expected value problem using decomposition, then continues and solves the stochastic universe problem by enumerating all possible combinations of realizations of second-stage random parameters.
- istrat = 6 Solves the stochastic problem using crude Monte Carlo sampling. No variance reduction technique is applied. This strategy is especially useful if you want to test a solution obtained by using the evaluation mode of DECIS. You have to specify the sample size used for the estimation. There is a maximum sample size DECIS can handle. However, this maximum sample size does not apply when using crude Monte Carlo. Therefore, in this mode you can specify very large sample sizes, which is useful when evaluating a particular solution.

istrat = 7 Refers to istrat = 1 plus istrat = 6. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using crude Monte Carlo sampling.

istrat = 8 Solves the stochastic problem using Monte Carlo pre-sampling. A Monte Carlo sample out of all possible universe scenarios, sampled from the original probability distribution, is taken, and the corresponding "sample problem" is solved using decomposition.

istrat = 9 Refers to istrat = 1 plus istrat = 8. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using Monte Carlo pre-sampling.

istrat = 10 Solves the stochastic problem using control variates. You also have to specify what approximation function and what sample size should be used for the estimation.

istrat = 11 Refers to istrat = 1 plus istrat = 10. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using control variates.

nsamples — Sample size used for the estimation. It should be set greater or equal to 30 in order to fulfill the assumption of large sample size used for the derivation of the probabilistic bounds. The default value is nsamples = 100.

nzrows — Number of rows reserved for cuts in the master problem. It specifies the maximum number of different cuts DECIS maintains during the course of the decomposition algorithm. DECIS adds one cut during each iteration. If the iteration count exceeds nzrows, then each new cut replaces a previously generated cut, where the cut is replaced that has the maximum slack in the solution of the (pseudo) master. If nzrows is specified as too small then DECIS may not be able to compute a solution and stops with an error message. If nzrows is specified as too large the solution time will increase. As an approximate rule set nzrows greater than or equal to the number of first-stage variables of the problem. The default value is nzrows = 100.

iwrite — Specifies whether the optimizer invoked for solving subproblems writes output or not. The default value is iwrite = 0.

iwrite = 0 No optimizer output is written.

iwrite = 1 Optimizer output is written to the file "MODEL.MO" in the case MINOS is used for solving subproblems or to the file MODEL.CPX in the case CPLEX is used for solving subproblems. The output level of the output can be specified using the optimizer options. It is intended as a debugging device. If you set iwrite = 1, for every master problem and for every subproblem solved the solution output is written. For large problems and large sample sizes the files"MODEL.MO" or "MODEL.CPX" may become very large, and the performance of DECIS may slow down.

ibug — Specifies the detail of debug output written by DECIS. The output is written to the file "MODEL.SCR", but can also be redirected to the screen by a separate parameter. The higher you set the number of ibug the more output DECIS will write. The parameter is intended to help debugging a problem and should be set to ibug = 0 for normal operation. For large problems and large sample sizes the file "MODEL.SCR" may become very large, and the performance of DECIS may slow down. The default value is ibug = 0.

ibug = 0 This is the setting for which DECIS does not write any debug output.

ibug = 1 In addition to the standard output, DECIS writes the solution of the master problem on each iteration of the Benders decomposition algorithm. Thereby it only writes out variable values which are nonzero. A threshold tolerance parameter for writing solution values can be specified, see below.

ibug = 2 In addition to the output of ibug = 1, DECIS writes the scenario index and the optimal objective value for each subproblem solved. In the case of solving the universe problem, DECIS also writes the probability of the corresponding scenario.

ibug = 3 In addition to the output of ibug = 2, DECIS writes information regarding importance sampling. In the case of using the additive approximation function, it reports the expected value for each i-th component of $\bar{\Gamma}_i$, the individual sample sizes N_i, and results from the estimation process. In the case of using the multiplicative approximation function it writes the expected value of the approximation function $\bar{\Gamma}$ and results from the estimation process.

ibug = 4 In addition to the output of ibug = 3, DECIS writes the optimal dual variables of the cuts on each iteration of the master problem.

ibug = 5 In addition to the output of ibug = 4, DECIS writes the coefficients and the right-hand side of the cuts on each iteration of the decomposition algorithm. In addition it checks if the cut computed is a support to the recourse function (or estimated recourse function) at the solution $\hat{x}^k$ at which it was generated. If it turns out that the cut is not a support, DECIS writes out the value of the (estimated) cut and the value of the (estimated) second stage cost at $\hat{x}^k$.

ibug = 6 In addition to the output of ibug = 5, DECIS writes a dump of the master problem and the subproblem in MPS format after having decomposed the problem specified in the core file. The dump of the master problem is written to the file "MODEL.P01" and the dump of the subproblem is written to the file "MODEL.P02". DECIS also writes a dump of the subproblem after the first iteration to the file "MODEL.S02".

iscratch — Specifies the internal unit number to which the standard and debug output is written. The default value is iscratch = 17, where the standard and debug output is written to the file "MODEL.SCR". Setting iscratch = 6 redirects the output to the screen. Other internal unit numbers could be used, e.g., the internal unit number of the printer, but this is not recommended.

ireg — Specifies whether or not DECIS uses regularized decomposition for solving the problem. This option is considered if MINOS is used as a master and subproblem solver, and is not considered if using CPLEX, since regularized decomposition uses a nonlinear term in the objective. The default value is ireg = 0.

rho — Specifies the value of the ρ parameter of the regularization term in the objective function. You will have to experiment to find out what value of rho works best for the problem you want to solve. There is no rule of thumb as to what value should be chosen. In many cases it has turned out that regularized decomposition reduces the iteration count if standard decomposition needs a large number of iterations. The default value is rho = 1000.

tolben — Specifies the tolerance for stopping the decomposition algorithm. The parameter is especially important for deterministic solution strategies, i.e., 1, 4, 5, 8, and 9. Choosing a very small value of tolben may result in a significantly increased number of iterations when solving the problem. The default value is 10^{-7}.

tolw — Specifies the nonzero tolerance when writing debug solution output. DECIS writes only variables whose values are nonzero, i.e., whose absolute optimal value is greater than or equal to tolw. The default value is 10^{-9}.

Example

In the following example the parameters istrat = 7, nsamples = 200, and nzrows = 200 are specified. All other parameters are set at their default values. DECIS first solves the expected value problem and then the stochastic problem using crude Monte Carlo sampling with a sample size of nsamples = 200. DECIS reserves space for a maximum of nzrows = 50 cuts.

```
7       "ISTRAT"
200     "NSAMPLES"
50      "NZROWS"
```

2.5.3. Setting MINOS parameters in the MINOS specification file

When you use MINOS as the optimizer for solving the master and the subproblems, you must specify optimization parameters in the MINOS specification file "MINOS.SPC". Each record of the file corresponds to the specification of one parameter and consists of a keyword and the value of the parameter in free format. Records having a "*" as their first character are considered as comment lines and are not further processed. For a detailed description of these parameters, see the MINOS Users' Guide (Murtagh and Saunders (1983) [5]. The following parameters should be specified with some consideration:

AIJ TOLERANCE — Specifies the nonzero tolerance for constraint matrix elements of the problem. Matrix elements a_{ij} that have a value for which $|a_{ij}|$ is less than "AIJ TOLERANCE" are considered by MINOS as zero and are automatically eliminated from the problem. It is wise to specify "AIJ TOLERANCE 0.0 "

SCALE — Specifies MINOS to scale the problem ("SCALE YES") or not ("SCALE NO"). It is wise to specify "SCALE NO".

ROWS — Specifies the number of rows in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ROWS" should be specified as the number of constraints in the core problem or greater.

COLUMNS — Specifies the number of columns in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "COLUMNS" should be specified as the number of variables in the core problem or greater.

ELEMENTS — Specifies the number of nonzero matrix coefficients in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ELEMENTS" should be specified as the number of nonzero matrix coefficients in the core problem or greater.

Example

The following example represents typical specifications for running DECIS with MINOS as the optimizer.

```
BEGIN SPECS
PRINT LEVEL                1
LOG FREQUENCY             10
SUMMARY FREQUENCY         10
MPS FILE                  12
ROWS                   20000
COLUMNS                50000
ELEMENTS              100000
ITERATIONS LIMIT       30000
*
FACTORIZATION FREQUENCY  100
AIJ TOLERANCE            0.0
*
SCALE                     NO
END OF SPECS
```

2.5.4. Setting CPLEX parameters using system environment variables

When you use CPLEX as the optimizer for solving the master and the subproblems, optimization parameters must be specified through system environment variables. You can specify the parameters "CPLEXLICDIR", "SCALELP", "NOPRESOLVE", "ITERLOG", "OPTIMALITYTOL", "FEASIBILIITYTOL", and "DUALSIMPLEX".

CPLEXLICDIR — Contains the path to the CPLEX license directory. For example, on an Unix system with the CPLEX license directory in /usr/users/cplex/cplexlicdir you issue the command *setenv CPLEXLICDIR /usr/users/cplex/cplexlicdir*.

SCALELP — Specifies CPLEX to scale the master and subproblems before solving them. If the environment variable is not set no scaling is used. Setting the environment variable, e.g., by issuing the command *setenv SCALELP yes*, scaling is switched on.

NOPRESOLVE — Allows to switch off CPLEX's presolver. If the environment variable is not set, presolve will be used. Setting the environment variable, e.g., by setting *setenv NOPRESOLVE yes*, no presolve will be used.

ITERLOG — Specifies the iteration log of the CPLEX iterations to be printed to the file "MODEL.CPX". If you do not set the environment variable no iteration log will be printed. Setting the environment variable, e.g., by setting *setenv ITERLOG yes*, the CPLEX iteration log is printed.

OPTIMALITYTOL — Specifies the optimality tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv OPTIMALITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.

FEASIBILIITYTOL — Specifies the feasibility tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv FEASIBILITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.

DUALSIMPLEX — Specifies the dual simplex algorithm of CPLEX to be used. If the environment variable is not set the primal simplex algorithm will be used. This is the default and works beautifully for most problems. If the environment variable is set, e.g., by setting *setenv DUALSIMPLEX yes*, CPLEX uses the dual simplex algorithm for solving both master and subproblems.

2.6. GAMS/DECIS output

After successfully having solved a problem, DECIS returns the objective, the optimal primal and optimal dual solution, the status of variables (if basic or not), and the status of equations (if binding or not) to GAMS. In the case of first-stage variables and equations you have all information in GAMS available as if you used any other solver, just instead of obtaining the optimal values for deterministic core problem you actually obtained the optimal values for the stochastic problem. However, for second-stage variables and constraints the expected values of the optimal primal and optimal dual solution are reported. This saves space and is useful for the calculation of risk measures. However, the information as to what the optimal primal and dual solutions were in the different scenarios of the stochastic programs is not reported back to GAMS. In a next release of the GAMS/DECIS interface the GAMS language is planned to be extended to being able to handle the scenario second-stage optimal primal and dual values at least for selected variables and equations.

While running DECIS outputs important information about the progress of the execution to your computer screen. After successfully having solved a problem, DECIS also outputs its optimal solution into the solution output file "MODEL.SOL". The debug output file "MODEL.SCR" contains important information about the optimization run, and the optimizer output files "MODEL.MO" (when using DECIS with MINOS) or "MODEL.CPX" (when using DECIS with CPLEX) contain solution output from the optimizer used. In the DECIS User's Guide you find a detailed discussion of how to interpret the screen output, the solution report and the information in the output files.

2.6.1. The screen output

The output to the screen allows you to observe the progress in the execution of a DECIS run. After the program logo and the copyright statement, you see four columns of output beeing written to the screen as long as the program proceeds. The first column (from left to right) represents the iteration count, the second column the lower bound (the optimal objective of the master problem), the third column the best upper bound (exact value or estimate of the total expected cost of the best solution found so far), and the fourth column the current upper bound (exact value or estimate of the total expected cost of current solution). After successful completion, DECIS quits with "Normal Exit", otherwise, if an error has been encountered, the programs stops with the message "Error Exit".

Example

When solving the illustrative example APL1P using strategy 5, we obtain the following report on the screen:

```
 T H E   D  E  C  I  S   S Y S T E M
 Copyright (c) 1989  -- 1999 by Dr. Gerd Infanger
 All rights reserved.

iter                lower           best upper        current upper

   0          -0.9935E+06
   1          -0.4626E+06           0.2590E+05           0.2590E+05
   2           0.2111E+05           0.2590E+05           0.5487E+06
   3           0.2170E+05           0.2590E+05           0.2697E+05
   4           0.2368E+05           0.2384E+05           0.2384E+05
   5           0.2370E+05           0.2384E+05           0.2401E+05
   6           0.2370E+05           0.2370E+05           0.2370E+05

iter                lower           best upper        current upper

   6           0.2370E+05
   7           0.2403E+05           0.2470E+05           0.2470E+05
   8           0.2433E+05           0.2470E+05           0.2694E+05
   9           0.2441E+05           0.2470E+05           0.2602E+05
  10           0.2453E+05           0.2470E+05           0.2499E+05
  11           0.2455E+05           0.2470E+05           0.2483E+05
  12           0.2461E+05           0.2467E+05           0.2467E+05
  13           0.2461E+05           0.2467E+05           0.2469E+05
  14           0.2461E+05           0.2465E+05           0.2465E+05
  15           0.2463E+05           0.2465E+05           0.2467E+05
  16           0.2463E+05           0.2465E+05           0.2465E+05
  17           0.2464E+05           0.2465E+05           0.2465E+05
  18           0.2464E+05           0.2464E+05           0.2464E+05
  19           0.2464E+05           0.2464E+05           0.2464E+05
  20           0.2464E+05           0.2464E+05           0.2464E+05
  21           0.2464E+05           0.2464E+05           0.2464E+05
  22           0.2464E+05           0.2464E+05           0.2464E+05

Normal Exit
```

2.6.2. The solution output file

The solution output file contains the solution report from the DECIS run. Its name is "MODEL.SOL". The file contains the best objective function value found, the corresponding values of the first-stage variables, the corresponding optimal second-stage cost, and a lower and an upper bound on the optimal objective of the problem. In addition, the number of universe scenarios and the settings for the stopping tolerance are reported. In the case of using a deterministic strategy for solving the problem, exact values are reported. When using Monte Carlo sampling, estimated values, their variances, and the sample size used for the estimation are reported. Instead of exact upper and lower bounds, probabilistic upper and lower bounds, and a 95% confidence interval, within which the true optimal solution lies with 95% confidence, are reported. A detailed description of the solution output file can be found in the DECIS User's Guide.

2.6.3. The debug output file

The debug output file contains the standard output of a run of DECIS containing important information about the problem, its parameters, and its solution. It also contains any error messages that may occur during a run of DECIS. In the case that DECIS does not complete a run successfully, the cause of the trouble can usually be located using the information in the debug output file. If the standard output does not give enough information you can set the debug parameter ibug in the parameter input file to a higher value and obtain additional debug output. A detailed description of the debug output file can be found in the DECIS User's Guide.

2.6.4. The optimizer output files

The optimizer output file "MODEL.MO" contains all the output from MINOS when called as a subroutine by DECIS. You can specify what degree of detail should be outputted by setting the appropriate "PRINT LEVEL" in the MINOS specification file. The optimizer output file "MODEL.CPX" reports messages and the iteration log (if switchwd on using the environment variable) from CPLEX when solving master and sub problems.

A. GAMS/DECIS illustrative Examples

A.1. Example APL1P

```
*   APL1P test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
             h    m    l
   g1        4.3  2.0  0.5
   g2        8.7  4.0  1.0;

parameter d(dl) demand                   / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost            total cost;
positive variable x(g)         capacity of generators;
positive variable y(g, dl)     operating level;
positive variable s(dl)        unserved demand;

equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;

cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
```

```
set stoch /out, pro /;

set omega1 / o11, o12, o13, o14 /;
table v1(stoch, omega1)
      o11   o12   o13   o14
out  -1.0  -0.9  -0.5  -0.1
pro   0.2   0.3   0.4   0.1
;

set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21   o22   o23   o24   o25
out  -1.0  -0.9  -0.7  -0.1  -0.0
pro   0.1   0.2   0.5   0.1   0.1
;


set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;


set omega4 / o41, o42, o43, o44 /;
table v4(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

* defining distributions
file stg /MODEL.STG/;
put stg;
put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1   ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
);
put "*" /;
loop(omega2,
put "x g2  omax g2   ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS demand h   ", v3("out", omega3),  " period2 ", v3("pro", omega3) /;
);
put "*" /;
```

```
loop(omega4,
put "RHS demand m   ", v4("out", omega4),  " period2 ", v4("pro", omega4) /;
);
put "*" /;
loop(omega5,
put "RHS demand l   ", v5("out", omega5),  " period2 ", v5("pro", omega5) /;
);
putclose stg;


* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

A.2. Example APL1PCA

```
*   APL1PCA test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
             h    m    l
   g1        4.3  2.0  0.5
   g2        8.7  4.0  1.0;

parameter d(dl) demand                   / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost            total cost;
positive variable x(g)         capacity of generators;
positive variable y(g, dl)     operating level;
positive variable s(dl)        unserved demand;

equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;

cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;


* defining independent stochastic parameters
set stoch /out, pro/;
```

```
set omega1 / o11, o12 /;
table v1(stoch,omega1)
       o11  o12
out    2.1  1.0
pro    0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
       o21  o22
out    2.0  1.0
pro    0.2  0.8 ;

parameter hm1(dl) / h  300., m  400., l  200. /;
parameter hm2(dl) / h  100., m  150., l  300. /;

* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL  v1   period2 ", v1("pro", omega1)/;
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2   period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

B. Error messages

1. ERROR in MODEL.STO: kwd, word1, word2 was not matched in first realization of block
   The specification of the stochastic parameters is incorrect. The stochastic parameter has not been specified in the specification of the first outcome of the block. When specifying the first outcome of a block always include all stochastic parameters corresponding to the block.

2. Option word1 word2 not supported
   You specified an input distribution in the stochastic file that is not supported. Check the DECIS manual for supported distributions.

3. Error in time file
   The time file is not correct. Check the file MODEL.TIM. Check the DECIS manual for the form of the time file.

4. ERROR in MODEL.STO: stochastic RHS for objective, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the objective row (row name2). Check file MODEL.STO.

5. ERROR in MODEL.STO: stochastic RHS in master, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the master problem (row name2). Check file MODEL.STO.

6. ERROR in MODEL.STO: col not found, name1
   The specification in the stochastic file is incorrect. The entry in the stochastic file, name1, is not found in the core file. Check file MODEL.STO.

7. ERROR in MODEL.STO: invalid col/row combination, (name1/name2)
   The stochastic file (MODEL.STO) contains an incorrect specification.

8. ERROR in MODEL.STO: no nonzero found (in B or D matrix) for col/row (name1, name2)
   There is no nonzero entry for the combination of name1 (col) and name2(row) in the B-matrix or in the D-matrix. Check the corresponding entry in the stochastic file (MODEL.STO). You may want to include a nonzero coefficient for (col/row) in the core file (MODEL.COR).

9. ERROR in MODEL.STO: col not found, name2
   The column name you specified in the stochastic file (MODEL.STO) does not exist in the core file (MODEL.COR). Check the file MODEL.STO.

10. ERROR in MODEL.STO: stochastic bound in master, col name2
    You specified a stochastic bound on first-stage variable name2. Check file MODEL.STO.

11. ERROR in MODEL.STO: invalid bound type (kwd) for col name2
    The bound type, kwd, you specified is invalid. Check file MODEL.STO.

12. ERROR in MODEL.STO: row not found, name2
    The specification in the stochastic file is incorrect. The row name, name2, does not exist in the core file. Check file MODEL.STO.

13. ERROR: problem infeasible
    The problem solved (master- or subproblem) turned out to be infeasible. If a subproblem is infeasible, you did not specify the problem as having the property of "complete recourse". Complete recourse means that whatever first-stage decision is passed to a subproblem, the subproblem will have a feasible solution. It is the best way to specify a problem, especially if you use a sampling based solution strategy. If DECIS encounters a feasible subproblem, it adds a feasibility cut and continues the execution. If DECIS encounters an infeasible master problem, the problem you specified is infeasible, and DECIS terminates. Check the problem formulation.

14. ERROR: problem unbounded
    The problem solved (master- or subproblem) turned out to be unbounded. Check the problem formulation.

15. ERROR: error code: inform
    The solver returned with an error code from solving the problem (master- or subproblem). Consult the users' manual of the solver (MINOS or CPLEX) for the meaning of the error code, inform. Check the problem formulation.

16. ERROR: while reading SPECS file
    The MINOS specification file (MINOS.SPC) containes an error. Check the specification file. Consult the MINOS user's manual.

17. ERROR: reading mps file, mpsfile
    The core file mpsfile (i.e., MODEL.COR) is incorect. Consult the DECIS manual for instructions regarding the MPS format.

18. ERROR: row 1 of problem ip is not a free row
    The first row of the problem is not a free row (i.e., is not the objective row). In order to make the fist row a free row, set the row type to be 'N'. Consult the DECIS manual for the MPS specification of the problem.

19. ERROR: name not found = nam1, nam2
    There is an error in the core file (MODEL.COR). The problem cannot be decomposed correctly. Check the core file and check the model formulation.

20. ERROR: matrix not in staircase form
    The constraint matrix of the problem as specified in core file (MODEL.COR) is not in staircase form. The first-stage rows and columns and the second-stage rows and columns are mixed within each other. Check the DECIS manual as to how to specify the core file. Check the core file and change the order of rows and columns.

References

[1] Brooke, A., Kendrik, D. and Meeraus, A. (1988): *GAMS, A Users Guide*, The Scientific Press, South San Francisco, California.

[2] CPLEX Optimization, Inc. (1989–1997): *Using the CPLEX Callable Library*, 930 Tahoe Blvd. Bldg. 802, Suite 279, Incline Village, NV 89451, USA.

[3] Infanger, G. (1994): *Planning Under Uncertainty – Solving Large-Scale Stochastic Linear Programs*, The Scientific Press Series, Boyd and Fraser.

[4] Infanger, G. (1997): *DECIS User's Guide*, Dr. Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002.

[5] Murtagh, B.A. and Saunders, M.A. (1983): MINOS User's Guide, SOL 83-20, Department of Operations Research, Stanford University, Stanford CA 94305.

DECIS License and Warranty

The software, which accompanies this license (the "Software") is the property of Gerd Infanger and is protected by copyright law. While Gerd Infanger continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum, which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may
    1. Use one copy of the Software on a single computer,
    2. Make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes,
    3. Use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network,
    4. After a written notice to Gerd Infanger, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement.
- You may not
    1. Copy the documentation, which accompanies the Software,
    2. Sublicense, rent or lease any portion of the Software,
    3. Reverse engineer, de-compile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software.

Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will he free from defects for a period of thirty (30) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of the warranty will be that Gerd Infanger will, at his option, replace any defective media returned to Gerd Infanger within the warranty period or refund the money you paid for the Software. Gerd Infanger does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL GERD INFANGER BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF GERD INFANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO CASE SHALL GERD INFANGER'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

General:

This Agreement will be governed by the laws of the State of California. This Agreement may only be modified by a license addendum, which accompanies this license or by a written document, which has been signed by both you and Gerd Infanger. Should you have any questions concerning this Agreement, or if you desire to contact Gerd Infanger for any reason, please write:

Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002, USA.

GAMS/DICOPT: A Discrete Continuous Optimization Package

IGNACIO E. GROSSMANN*
JAGADISAN VISWANATHAN* ALDO VECCHIETTI*
RAMESH RAMAN† ERWIN KALVELAGEN†

March 22, 2002

1 Introduction

DICOPT is a program for solving mixed-integer nonlinear programming (MINLP) problems that involve linear binary or integer variables and linear and nonlinear continuous variables. While the modeling and solution of these MINLP optimization problems has not yet reached the stage of maturity and reliability as linear, integer or non-linear programming modeling, these problems have a rich area of applications. For example, they often arise in engineering design, management sciences, and finance. DICOPT (DIscrete and Continuous OPTimizer) was developed by J. Viswanathan and Ignacio E. Grossmann at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside DICOPT solves a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under GAMS.

Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum.

The GAMS/DICOPT system has been designed with two main goals in mind:

- to build on existing modeling concepts and to introduce a minimum of extensions to the existing modeling language and provide upward compatibility to ensure easy transition from existing modeling applications to nonlinear mixed-integer formulations

*Engineering Research Design Center, Carnegie Mellon University, Pittsburgh, PA
†GAMS Development Corporation, Washington D.C.

- to use existing optimizers to solve the DICOPT sub-problems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MIP solvers become automatically and immediate available to DICOPT.

2 Requirements

In order to use DICOPT you will need to have access to a licensed GAMS BASE system as well as at least one licensed MIP solver and one licensed NLP solver. For difficult models it is advised to have access to multiple solvers. Free student/demo systems are available from GAMS Development Corporation. These systems are restricted in the size of models that can be solved.

3 How to run a model with GAMS/DICOPT

DICOPT is capable of solving only MINLP models. If you did not specify DICOPT as the default solver, then you can use the following statement in your GAMS model:

```
option minlp = dicopt;
```

It should appear before the solve statement. DICOPT automatically uses the default MIP and NLP solver to solve its sub-problems. One can override this with the GAMS statements like:

```
option nlp = conopt2; { or any other nlp solver }
option mip = cplex;   { or any other mip solver }
```

These options can also be specified on the command line, like:

```
> gams mymodel minlp=dicopt nlp=conopt2 mip=cplex
```

In the IDE (Integrated Development Enviroment) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include `minos5`, `minos`, `conopt`, `conopt2`, and `snopt`. Possible MIP solvers are `cplex`, `osl`, `osl2`, `osl3`, `xpress`, and `xa`.

With an option file it is even possible to use alternate solvers in different cycles. Section 8 explains this is in detail.

4 Overview of DICOPT

DICOPT solves models of the form:

MINLP	min or max	$f(x,y)$
	subject to	$g(x,y) \sim b$
		$\ell_x \leq x \leq u_x$
		$y \in \lceil \ell_y \rceil, ..., \lfloor u_y \rfloor$

where x are the continuous variables and y are the discrete variables. The symbol $\sim$ is used to denote a vector of relational operators $\{\leq, =, \geq\}$. The constraints can be either linear or non-linear. Bounds ℓ and u on the variables are handled directly. $\lceil x \rceil$ indicates the smallest integer, greater than or equal to x. Similarly, $\lfloor x \rfloor$ indicates the largest integer, less than or equal to x. The discrete variables can be either integer variables or binary variables.

5 The algorithm

The algorithm in DICOPT is based on three key ideas:

- Outer Approximation
- Equality Relaxation
- Augmented Penalty

Outer Approximation refers to the fact that the surface described by a convex function lies above the tangent hyper-plane at any interior point of the surface. (In 1-dimension, the analogous geometrical result is that the tangent to a convex function at an interior point lies below the curve). In the algorithm outer-approximations are attained by generating linearizations at each iterations and accumulating them in order to provide successively improved linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region.

Equality Relaxation is based on the following result from non-linear programming. Suppose the MINLP problem is formulated in the form:

$$\begin{aligned} \text{minimize or maximize } & f(x) + c^T y \\ \text{subject to } & G(x) + Hy \sim b \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{aligned} \tag{1}$$

i.e. the discrete variables are binary variables and they appear linearly in the model.

If we reorder the equations into equality and inequality equations, and convert the problem into a minimization problem, we can write:

$$
\begin{aligned}
&\text{minimize } c^T y + f(x) \\
&\text{subject to } Ay + h(x) = 0 \\
&\qquad\qquad By + g(x) \leq 0 \\
&\qquad\qquad \ell \leq x \leq u \\
&\qquad\qquad y \in \{0,1\}
\end{aligned}
\tag{2}
$$

Let $y^{(0)}$ be any fixed binary vector and let $x^{(0)}$ be the solution of the corresponding NLP subproblem:

$$
\begin{aligned}
&\text{minimize } c^T y^{(0)} + f(x) \\
&\text{subject to } Ay^{(0)} + h(x) = 0 \\
&\qquad\qquad By^{(0)} + g(x) \leq 0 \\
&\qquad\qquad \ell \leq x \leq u
\end{aligned}
\tag{3}
$$

Further let

$$
\begin{aligned}
T^{(0)} &= \text{diag}(t_{i,i}) \\
t_{i,i} &= \text{sign}(\lambda_i)
\end{aligned}
\tag{4}
$$

where λ_i is the Lagrange multiplier of the i-th equality constraint.

If f is pseudo-convex, h is quasi-convex, and g is quasi-convex, then x^0 is also the solution of the following NLP:

$$
\begin{aligned}
&\text{minimize } c^T y^{(0)} + f(x) \\
&\text{subject to } T^{(0)}(Ay^{(0)} + h(x)) \leq 0 \\
&\qquad\qquad By^{(0)} + g(x) \leq 0 \\
&\qquad\qquad \ell \leq x \leq u
\end{aligned}
\tag{5}
$$

In colloquial terms, under certain assumptions concerning the convexity of the nonlinear functions, an equality constraint can be "relaxed" to be an inequality constraint. This property is used in the MIP master problem to accumulate linear approximations.

Augmented Penalty refers to the introduction of (non-negative) slack variables on the right hand sides of the just described inequality constraints and the modification of the objective function when assumptions concerning convexity do not hold.

The algorithm underlying DICOPT starts by solving the NLP in which the 0-1 conditions on the binary variables are relaxed. If the solution to this problem yields an integer solution the search stops. Otherwise, it continues with an alternating sequence of nonlinear programs (NLP) called subproblems and

mixed-integer linear programs (MIP) called master problems. The NLP subproblems are solved for fixed 0-1 variables that are predicted by the MIP master problem at each (major) iteration. For the case of convex problems, the master problem also provides a lower bound on the objective function. This lower bound (in the case of minimization) increases monotonically as iterations proceed due to the accumulation of linear approximations. Note that in the case of maximization this bound is an upper bound. This bound can be used as a stopping criterion through a DICOPT option `stop 1` (see section 8). Another stopping criterion that tends to work very well in practice for non-convex problems (and even on convex problems) is based on the heuristic: stop as soon as the NLP subproblems start worsening (i.e. the current NLP subproblem has an optimal objective function that is worse than the previous NLP subproblem). This stopping criterion relies on the use of the augmented penatly and is used in the description of the algorithm below. This is also the default stopping criterion in the implementation of DICOPT. The algorithm can be stated briefly as follows:

1. Solve the NLP relaxation of the MINLP program. If $y^{(0)} = y$ is integer, stop("integer optimum found"). Else continue with step 2.

2. Find an integer point $y^{(1)}$ with an MIP master problem that features an augmented penalty function to find the minimum over the convex hull determined by the half-spaces at the solution $(x^{(0)}, y^{(0)})$.

3. Fix the binary variables $y = y^{(1)}$ and solve the resulting NLP. Let $(x^{(1)}, y^{(1)})$ be the corresponding solution.

4. Find an integer solution $y^{(2)}$ with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at $y^{(0)}$ and $y^{(1)}$.

5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e. half-spaces - at the new KKT point).

In the MIP problems integer cuts are added to the model to exclude previously determined integer vectors $y^{(1)}, y^{(2)}, ..., y^{(K)}$.

For a detailed description of the theory and references to earlier work, see [5, 3, 1].

The algorithm has been extended to handle general integer variables and integer variables appearing nonlinearly in the model.

6 Modeling

6.1 Relaxed model

Before solving a model with DICOPT, it is strongly advised to experiment with the relaxed model where the integer restrictions are ignored. This is the `RMINLP` model. As the DICOPT will start solving the relaxed problem and can use an existing relaxed optimal solution, it is a good idea to solve the `RMINLP` always before attempting to solve the `MINLP` model. I.e. the following fragment is not detrimental with respect to performance:

```
  model m /all/;
  option nlp=conopt2;
  option mip=cplex;
  option rminlp=conopt2;
  option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  abort$(m.modelstat > 2.5) "Relaxed model could not be solved";

*
* solve minlp model
*
  solve m using minlp minimizing z;
```

The second `SOLVE` statement will only be executed if the first `SOLVE` was succesful, i.e. if the model status was one (optimal) or two (locally optimal).

In general it is not a good idea to try to solve an MINLP model if the relaxed model can not be solved reliably. As the RMINLP model is a normal NLP model, some obvious points of attention are:

- Scaling. If a model is poorly scaled, an NLP solver may not be able find the optimal or even a feasible solution. Some NLP solvers have automatic scaling algorithms, but often it is better to attack this problem on the modeling level. The GAMS scaling facility can help in this respect.
- Starting point. If a poor starting point is used, the NLP solver may not be able to find a feasible or optimal solution. A starting point can be set by setting level values, e.g. `X.L = 1;`. The GAMS default levels are zero, with is often not a good choice.
- Adding bounds. Add bounds so that all functions can be properly evaluated. If you have a function $\sqrt{x}$ or $\log(x)$ in the model, you may want to add a bound `X.LO=0.001;`. If a function like $\log(f(x))$ is used, you may want to introduce an auxiliary variable and equation $y = f(x)$ with an appropriate bound `Y.LO=0.001;`.

In some cases the relaxed problem is the most difficult model. If you have more than one NLP solver available, you may want to try a sequence of them:

```
  model m /all/;
  option nlp=conopt2;
  option mip=cplex;
  option rminlp=conopt2;
  option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  if (m.modelstat > 2.5,
     option rminlp=minos;
     solve m using rminlp minimizing z;
  );
  if (m.modelstat > 2.5,
     option rminlp=snopt;
     solve m using rminlp minimizing z;
  );

*
* solve minlp model
*
  solve m using minlp minimizing z;
```

In this fragment, we first try to solve the relaxed model using `CONOPT2`. If that fails we try `MINOS`, and if that solve also fails, we try `SNOPT`.

It is worthwhile to spend some time in getting the relaxed model to solve reliably and speedily. In most cases, modeling improvements in the relaxed model, such as scaling, will also benefit the subsequent NLP sub-problems. In general these modeling improvements turn out to be rather solver independent: changes that improve the performance with CONOPT will also help solving the model with MINOS.

6.2 OPTCR and OPTCA

The DICOPT algorithm assumes that the integer sub-problems are solved to optimality. The GAMS options for `OPTCR` and `OPTCA` are therefore ignored: subproblems are solved with both tolerances set to zero. If you really want to solve a MIP sub-problem with an optimality tolerance, you can use the DICOPT option file to set `OPTCR` or `OPTCA` in there. For more information see section 8.

For models with many discrete variables, it may be necessary to introduce an `OPTCR` or `OPTCA` option in order to solve the model in acceptable time. For models with a limited number of integer variables the default to solve MIP sub-models to optimality may be acceptable.

6.3 Integer formulations

A number of MIP formulations are not very obvious and pose a demand on the modeler with respect to knowledge and experience. A good overview of integer programming modeling is given in [6].

Many integer formulations use a so-called big-M construct. It is important to choose small values for those big-M numbers. As an example consider the

fixed charge problem where $y_i \in \{0,1\}$ indicate if facility i is open or closed, and where x_i is the production at facility i. Then the cost function can be modeled as:

$$\begin{aligned} C_i &= f_i y_i + v_i x_i \\ x_i &\leq M_i y_i \\ y_i &\in \{0,1\} \\ 0 &\leq x_i \leq cap_i \end{aligned} \tag{6}$$

where f_i is the fixed cost and v_i the variables cost of operating facility i. In this case M_i should be chosen large enough that x_i is not restricted if $y_i = 1$. On the other hand, we want it as small as possible. This leads to the choice to have M_i equal to the (tight) upperbound of variable x_i (i.e. the capacity cap_i of facility i).

6.4 Non-smooth functions

NLP modelers are alerted by GAMS against the use of non-smooth functions such as `min()`, `max()`, `smin()`, `smax()` and `abs()`. In order to use these functions, a non-linear program needs to be declared as a DNLP model instead of a regular NLP model:

```
option dnlp=conopt2;
model m /all/;
solve m minimizing z using dnlp;
```

This construct is to warn the user that problems may arise due to the use of non-smooth functions.

A possible solution is to use a smooth approximation. For instance, the function $f(x) = |x|$ can be approximated by $g(x) = \sqrt{x^2 + \varepsilon}$ for some $\varepsilon > 0$. This approximation does not contain the point $(0,0)$. An alternative approximation can be devised that has this property:

$$f(x) \approx \frac{2x}{1 + e^{-x/h}} - x \tag{7}$$

For more information see [2].

For MINLP models, there is not such a protection against non-smooth functions. However, the use of such functions is just as problematic here. However, with MINLP models we have the possibility to use discrete variables, in order to model if-then-else situations. For the case of the absolute value for instance we can replace x by $x^+ - x^-$ and $|x|$ by $x^+ + x^-$ by using:

$$
\begin{aligned}
x &= x^+ - x^- \\
|x| &= x^+ + x^- \\
x^+ &\leq \delta M \\
x^- &\leq (1-\delta) M \\
x^+, x^- &\geq 0 \\
\delta &\in \{0,1\}
\end{aligned}
\tag{8}
$$

where δ is a binary variable.

7 GAMS Options

GAMS options are specified in the GAMS model source, either using the `option` statement or using a model suffix.

7.1 The OPTION statement

An option statement sets a global parameter. An option statement should appear *before* the `solve` statement, as in:

```
model m /all/;
option iterlim=100;
solve m using minlp minimizing z;
```

Here follows a list of option statements that affect the behavior of DICOPT:

option domlim = n**;**

This option sets a limit on the total accumulated number of non-linear function evaluation errors that are allowed while solving the NLP subproblems or inside DICOPT itself. An example of a function evaluation error or domain error is taking the square root of a negative number. This situations can be prevented by adding proper bounds. The default is zero, i.e. no function evaluation errors are allowed.

In case a domain error occurs, the listing file will contain an appropriate message, including the equation that is causing the problem, for instance:

```
**** ERRORS(S) IN EQUATION loss(cc,sw)
     2 instance(s) of - UNDEFINED REAL POWER (RETURNED  0.0E+00)
```

If such errors appear you can increase the `DOMLIM` limit, but often it is better to prevent the errors to occur. In many cases this can be accomplished by adding appropriate bounds. Sometimes you will need to add extra variables and equations to accomplish this. For instance with an expression like $\log(x-y)$, you may want to introduce a variable $z > \varepsilon$ and an equation $z = x - y$, so that the expression can be rewritten as $\log(z)$.

option iterlim = n;
: This option sets a limit on the total accumulated (minor) iterations performed in the MIP and NLP subproblems. The default is 1000.

option minlp = dicopt;
: Selects DICOPT to solve MINLP problems.

option mip = s;
: This option sets the MIP solver to be used for the MIP master problems. Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

option nlp = s;
: This option sets the NLP solver to be used for the NLP sub-problems. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

option optca = x;
: This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

option optcr = x;
: This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

option reslim = x;
: This option sets a limit on the total accumulated time (in seconds) spent inside DICOPT and the subsolvers. The default is 1000 seconds.

option sysout = on;
: This option will print extra information to the listing file.

In the list above (and in the following) n indicates an integer number. GAMS will also accept fractional values: they will be rounded. Options marked with an x parameter expect a real number. Options with an s parameter, expect a string argument.

7.2 The model suffix

Some options are set by assigning a value to a model suffix, as in:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

Here follows a list of model suffices that affect the behavoir of DICOPT:

***m*.dictfile = 1;**
: This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). This information is needed when the DICOPT option `nlptracelevel` is used. Otherwise this option can be ignored.

***m*.iterlim = *n*;**

Sets the total accumulated (minor) iteration limit. This option overrides the global iteration limit set by an option statement. E.g.,

```
model m /all/;
m.iterlim = 100;
option iterlim = 1000;
solve m using minlp minimizing z;
```

will cause DICOPT to use an iteration limit of 100.

***m*.optfile = 1;**

This option instructs DICOPT to read an option file `dicopt.opt`. This file should be located in the current directory (or the project directory when using the GAMS IDE). The contents of the option file will be echoed to the listing file and to the screen (the log file):

```
--- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
> maxcycles 10
--- DICOPT: Starting major iteration 1
```

If the option file does not exist, the algorithm will proceed using its default settings. An appropriate message will be displayed in the listing file and in the log file:

```
--- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
--- DICOPT: File does not exist, using defaults...
--- DICOPT: Starting major iteration 1
```

***m*.optfile = *n*;**

If $n > 1$ then the option file that is read is called `dicopt.op`n (for $n = 2, ..., 9$) or `dicopt.o`n (for $n = 10, ..., 99$). E.g. `m.optfile=2;` will cause DICOPT to read `dicop.op2`.

***m*.prioropt = 1;**

This option will turn on the use of priorities on the discrete variables. Priorities influence the branching order chosen by the MIP solver during solution of the MIP master problems. The use of priorities can greatly impact the performance of the MIP solver. The priorities themselves have to be specified using the `.prior` variables suffix, e.g. `x.prior(i,j) = ord(i);`. Contrary to intuition, variables with a lower value for their priority are branched on before variables with a higher priority. I.e. the most important variables should get lower priority values.

***m*.reslim = *x*;**

Sets the total accumulated time limit. This option overrides the global time limit set by an option statement.

8 DICOPT options

This sections describes the options that can be specified in the DICOPT option file. This file is usually called `dicopt.opt`. In order to tell DICOPT to read this file, you will need to set the `optfile` model suffix, as in:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in case the IDE (Integrated Development Environment) is used, in the project directory.

The option file is a standard text file, with a single option on each line. All options are case-insensitive. A line is a comment line if it starts with an asterisk, `*`, in column one. A valid option file can look like:

```
* stop only on infeasible MIP or hitting a limit
stop 0
* use minos to solve first NLP sub problem
* and conopt2 for all subsequent ones
nlpsolver minos conopt2
```

Here follows a list of available DICOPT options:

continue n

This option can be used to let DICOPT continue in case of NLP solver failures. The preferred approach is to fix the model, such that NLP subproblems solve without problems. However, in some cases we can ignore (partial) failures of an NLP solver in solving the NLP subproblems as DICOPT may recover later on. During model debugging, you may therefore add the option `continue 0`, in order for DICOPT to function in a more finicky way.

continue 0

Stop on solver failure. DICOPT will terminate when an NLP subproblem can not be solved to optimality. Some NLP solvers terminate with a status other than optimal if not all of the termination criteria are met. For instance, the change in the objective function is negligable (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close the (local) optimum. Using `continue 0` will cause DICOPT not to accept such a solution.

continue 1

NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver can not make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option.

continue 2

NLP subproblem failures resulting in a non-optimal but feasible solution are accepted (as in option `continue 1`). NLP subproblem failures resulting in an infeasible solution are ignored. The corresponding configuration of discrete variables is forbidden to be used again. An integer cut to accomplish this, is added to subsequent MIP master problems. Note that the relaxed NLP solution should be feasible. This setting is the default.

domlim $i_1\ i_2\ \ldots\ i_n$

Sets a limit of the number of function and derivative evaluation errors for a particular cycle. A number of -1 means that the global GAMS option `domlim` is used. The last number i_n sets a domain error limit for all cycles $n, n+1, \ldots$.

Example: `domlim 0 100 0`

The NLP solver in the second cycle is allowed to make up to 100 evaluation errors, while all other cycles must be solved without evaluation errors.

The default is to use the global GAMS `domlim` option.

epsmip x

This option can be used to relax the test on MIP objective functions. The objective function values of the MIP master problems should form a monotonic worsening curve. This is not the case if the MIP master problems are not solved to optimality. Thus, if the options `OPTCR` or `OPTCA` are set to a nonzero value, this test is bypassed. If the test fails, DICOPT will fail with a message:

```
The MIP solution became better after adding integer cuts.
Something is wrong. Please check if your model is properly
scaled. Also check your big M formulations -- the value
of M should be relatively small.

This error can also occur if you used a MIP solver option
file with a nonzero OPTCR or OPTCA setting. In that case
you may want to increase the EPSMIP setting using a
DICOPT option file.
```

The value of

$$\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|} \tag{9}$$

is compared against `epsmip`. In case the test fails, but you want DICOPT to continue anyway, you may want to increase the value of `epsmip`. The current values used in the test (previous and current MIP objective, `epsmip`) are printed along with the message above, so you will have information about how much you should increase `epsmip` to pass the test. Normally, you should not have to change this value. The default is $x = 1.0e{-}6$.

epsx x

This tolerance is used to distinguish integer variables that are set to an integer value by the user, or integer variables that are fractional. See the option `relaxed`. Default: $x = 1.0e - 3$.

keepfiles

This option is for debugging purposes. By default DICOPT will remove any scratch files that are written. With this option, you can tell DICOPT not to delete those temporary files. This option is especially useful in combination with the GAMS main driver program `gamskeep`, which will not delete the GAMS scratch directory after the job has finished.

maxcycles n

The maximum number of cycles or major iterations performed by DICOPT. The default is $n = 20$.

mipiterlim $i_1\ i_2\ \ldots\ i_n$

Sets an iteration limit on individual MIP master problems. The last number i_n is valid for all subsequent cycles $n, n+1, \ldots$. A number of -1 indicates that there is no (individual) limit on the corresponding MIP master problem. A global iteration limit is maintained through the GAMS option `iterlim`.

Example: `mipiterlim 10000 -1`

The first MIP master problem can not use more than 10000 iterations, while subsequent MIP master problems are not individually restricted.

Example: `mipiterlim 10000`

Sets an iteration limit of 10000 on all MIP master problems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict iteration counts on individual solves of MIP master problems.

mipoptfile $s_1\ s_2\ \ldots\ s_n$

Specifies the option file to be used for the MIP master problems. Several option files can be specified, separated by a blank. If a digit `1` is entered, the default option file for the MIP solver in question is being used. The digit `0` indicates: no option file is to be used. The last option file is also used for subsequent MIP master problems.

Example: `mipoptfile mip.opt mip2.opt 0`

This option will cause the first MIP master problem solver to read the option file `mip.opt`, the second one to read the option file `mip2.opt` and subsequent MIP master problem solvers will not use any option file.

Example: `mipoptfile 1`
This will cause the MIP solver for all MIP subproblems to read a default option file (e.g. `cplex.opt`, `xpress.opt`, `osl2.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

mipreslim $x_1\ x_2\ \ldots\ x_n$
Sets a resource (time) limit on individual MIP master problems. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$. A number -1.0 means that the corresponding MIP master problem is not individually time restricted. A global time limit is maintained through the GAMS option `reslim`.

Example: `mipreslim -1 10000 -1`
The MIP master problem in cycle 2 can not use more than 100 seconds, while subsequent MIP master problems are not individually restricted.

Example: `mipreslim 1000`
Sets a time limit on all MIP master problems of 1000 seconds.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict individually the time a solver can spent on the MIP master problem.

mipsolver $s_1\ s_2\ \ldots\ s_n$
This option specifies with MIP solver to use for the MIP master problems.

Example: `mipsolver cplex osl2`
This instructs DICOPT to use Cplex for the first MIP and OSL2 for the second and subsequent MIP problems. The last entry may be used for more than one problem.

The names to be used for the solvers are the same as one uses in the GAMS statement `OPTION MIP=....;`. The default is to use the default MIP solver.
Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

nlpiterlim $i_1\ i_2\ \ldots\ i_n$
Sets an iteration limit on individual NLP subproblems. The last number i_n is valid for all subsequent cycles $n, n+1, \ldots$. A number of -1 indicates that there is no (individual) limit on the corresponding NLP subproblem. A global iteration limit is maintained through the GAMS option `iterlim`.

Example: `nlpiterlim 1000 -1`
The first (relaxed) NLP subproblem can not use more than 1000 iterations, while subsequent NLP subproblems are not individually restricted.

Example: `nlpiterlim 1000`
Sets an iteration limit of 1000 on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default is not to restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

nlpoptfile s_1 s_2 ... s_n
Specifies the option file to be used for the NLP subproblems. Several option files can be specified, separated by a blank. If a digit `1` is entered, the default option file for the NLP solver in question is being used. The digit `0` indicates: no option file is to be used. The last option file is also used for subsequent NLP subproblems.

Example: `nlpoptfile nlp.opt nlp2.opt 0`
This option will cause the first NLP subproblem solver to read the option file `nlp.opt`, the second one to read the option file `nlp2.opt` and subsequent NLP subproblem solvers will not use any option file.

Example: `nlpoptfile 1`
This will cause the NLP solver for all NLP subproblems to read a default option file (e.g. `conopt2.opt`, `minos.opt`, `snopt.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

nlpreslim x_1 x_2 ... x_n
Sets a resource (time) limit on individual NLP subproblems. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$. A number -1.0 means that the corresponding NLP subproblem is not individually time restricted. A global time limit is maintained through the GAMS option `reslim`.

Example: `nlpreslim 100 -1`
The first (relaxed) NLP subproblem can not use more than 100 seconds, while subsequent NLP subproblems are not individually restricted.

Example: `nlpreslim 1000`
Sets a time limit of 1000 seconds on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict individually the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

nlpsolver s_1 s_2 ... s_n
This option specifies which NLP solver to use for the NLP subproblems.

Example: `nlpsolver conopt2 minos snopt`
tells DICOPT to use CONOPT2 for the relaxed NLP, MINOS for the second NLP subproblem and SNOPT for the third and subsequent ones. The last entry is used for more than one subproblem: for all subsequent ones DICOPT will use the last specified solver.

The names to be used for the solvers are the same as one uses in the GAMS statement `OPTION NLP=....;`. The default is to use the default NLP solver. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

nlptracefile *s*
Name of the files written if the option `nlptracelevel` is set. Only the stem is needed: if the name is specified as `nlptracefile nlptrace`, then files of the form `nlptrace.001`, `nlptrace.002`, etc. are written. These files contain the settings of the integer variables so that NLP subproblems can be investigated independently of DICOPT. Default: `nlptrace`.

nlptracelevel *n*
This sets the level for NLP tracing, which writes a file for each NLP sub-problem, so that NLP sub-problems can be investigated outside the DICOPT environment. See also the option `nlptracefile`.

nlptracelevel 0
No trace files are written. This is the default.

nlptracelevel 1
A GAMS file for each NLP subproblem is written which fixes the discrete variables.

nlptracelevel 2
As `nlptracelevel 1`, but in addition level values of the continuous variables are written.

nlptracelevel 3
As `nlptracelevel 2`, but in addition marginal values for the equations and variables are written.

By including a trace file to your original problem, and changing it into an MINLP problem, the subproblem will be solved directly by an NLP solver. This option only works if the names in the model (names of variables and equations) are exported by GAMS. This can be accomplished by using the *m*.`dictfile` model suffix, as in `m.dictfile=1;`. In general it is more convenient to use the `CONVERT` solver to generate isolated NLP models (see section 10.4).

optca $x_1\ x_2\ \ldots\ x_n$
The absolute optimality criterion for the MIP master problems. The GAMS option `optca` is ignored, as by default DICOPT wants to solve MIP master problems to optimality. To allow to solve large problem, it is

possible to stop the MIP solver earlier, by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optca`, the MIP solver is instructed to stop as soon as the gap between the best possible integer solution and the best found integer solution is less than x, i.e. stop as soon as

$$|\text{BestFound} - \text{BestPossible}| \leq x \tag{10}$$

It is possible to specify a different `optca` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$.

Example: `optca 10`
Stop the search in all MIP problems as soon as the absolute gap is less than 10.

Example: `optca 0 10 0`
Sets a nonzero `optca` value of 10 for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

optcr $x_1\ x_2\ \ldots\ x_n$
The relative optimality criterion for the MIP master problems. The GAMS option `optca` is ignored, as by default DICOPT wants to solve MIP master problems to optimality. To allow to solve large problem, it is possible to stop the MIP solver earlier, by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optcr`, the MIP solver is instructed to stop as soon as the relative gap between the best possible integer solution and the best found integer solution is less than x, i.e. stop as soon as

$$\frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \leq x \tag{11}$$

Note that the relative gap can not be evaluated if the best possible integer solution is zero. In those cases the absolute optimality criterion `optca` can be used. It is possible to specify a different `optcr` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$.

Example: `optcr 0.1`
Stop the search in all the MIP problems as soon as the relative gap is smaller than 10%.

Example: `optcr 0 0.01 0`
Sets a nonzero `optcr` value of 1% for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

relaxed n
In some cases it may be possible to use a known configuration of the discrete variables. Some users have very difficult problems, where the relaxed problem can not be solved, but where NLP sub-problems with the

integer variables fixed are much easier. In such a case, if a reasonable integer configuration is known in advance, we can bypass the relaxed NLP and tell DICOPT to directly start with this integer configuration. The integer variables need to be specified by the user before the solve statement by assigning values to the levels, as in `Y.L(I) = INITVAL(I);`.

relaxed 0
: The first NLP sub-problem will be executed with all integer variables fixed to the values specified by the user. If you don't assign a value to an integer variable, it will retain it's current value, which is zero by default.

relaxed 1
: The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.

relaxed 2
: The first NLP subproblem will be executed with some variables fixed and some relaxed. The program distinguishes the fixed from the relaxed variables by comparing the initial values against the bounds and the tolerance allowed `EPSX`. `EPSX` has a default value of 1.e-3. This can be changed in through the option file.

stop *n*
: This option defines the stopping criterion to be used. The search is always stopped when the (minor) iteration limit (the `iterlim` option), the resource limit (the `reslim` option), or the major iteration limit (see *maxcycles*) is hit or when the MIP master problem becomes infeasible.

stop 0
: Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will find often the optimal solution using one of the more optimistic stopping rules.

stop 1
: Stop as soon as the bound defined by the objective of the last MIP master problem is worse than the best NLP solution found (a "crossover" occurred). For convex problems this gives a global solution, provided the weights are large enough. This stopping criterion should only be used if it is known or it is very likely that the nonlinear functions are convex. In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut-off with the setting `stop 1`.

stop 2
: Stop as soon as the NLP subproblems stop to improve. This"worsening"

criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic works often very well. Even on convex problems, in many cases it terminates the search very early while providing an optimal or a very good integer solution. The criterion is not checked before major iteration three.

stop 3
Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).

Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section VI shows some experiments with these stopping criteria.

weight x
The value of the penalty coefficients. Default $x = 1000.0$.

9 DICOPT output

DICOPT generates lots of output on the screen. Not only does DICOPT itself writes messages to the screen, but also the NLP and MIP solvers that handle the sub-problems. The most important part is the last part of the screen output.

In this section we will discuss the output that DICOPT writes to the screen and the listing file using the model `procsel.gms` (this model is part of the GAMS model library). A DICOPT log is written there and the reason why DICOPT terminated.

```
--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.05        8      0      conopt2
  MIP    1         2.48869      0.28        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt2
  MIP    2         2.17864      0.22       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt2
  MIP    3         1.42129      0.22       12      0      cplex
  NLP    4         1.41100      0.00        8      0      conopt2
--- DICOPT: Terminating...
--- DICOPT: Stopped on NLP worsening

      The search was stopped because the objective function
      of the NLP subproblems started to deteriorate.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSEL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion
```

Notice that the integer solutions are provided by the NLP's except for major iteration one (the first NLP is the relaxed NLP). For all NLP's except the relaxed one, the binary variables are fixed, according to a pattern determined by the previous MIP which operates on a linearized model. The integer solutions marked with a '<' are an improvement. We see that the NLP in cycle 4 starts to deteriorate, and DICOPT stops based on its default stopping rule.

It should be noted that if the criterion `stop 1` had been used the search would have been terminated at iteration 3. The reason is that the upper bound to the profit predicted by the MIP (1.42129) exceeds the best current NLP solution (1.9231). Since it can be shown that the MINLP involves convex nonlinear functions, 1.9231 is the global optimum and the criterion `stop 1` is rigorous.

A similar output can be found in the listing file:

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

     MODEL   process             OBJECTIVE  pr
     TYPE    MINLP               DIRECTION  MAXIMIZE
     SOLVER  DICOPT              FROM LINE  98

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      8 INTEGER SOLUTION
**** OBJECTIVE VALUE                1.9231

 RESOURCE USAGE, LIMIT          0.771      1000.000
 ITERATION COUNT, LIMIT        51         10000
 EVALUATION ERRORS              0             0

--- DICOPT: Stopped on NLP worsening

      The search was stopped because the objective function
      of the NLP subproblems started to deteriorate.

 ------------------------------------------------------------------
 Dicopt2x-C    Jul  4, 2001 WIN.DI.DI 20.1 026.020.039.WAT
 ------------------------------------------------------------------

     Aldo Vecchietti and Ignacio E. Grossmann
     Engineering Design Research Center
     Carnegie Mellon University
     Pittsburgh, Pennsylvania 15213

     Erwin Kalvelagen
     GAMS Development Corp.
     1217 Potomac Street, N.W.
     Washington DC 20007
 ------------------------------------------------------------------
 DICOPT Log File
 ------------------------------------------------------------------
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.05        8      0      conopt2
  MIP    1         2.48869      0.28        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt2
  MIP    2         2.17864      0.22       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt2
```

```
 MIP    3         1.42129      0.22       12      0      cplex
 NLP    4         1.41100      0.00        8      0      conopt2
----------------------------------------------------------------------
  Total solver times :  NLP =     0.05   MIP =     0.72
  Perc. of total     :  NLP =     6.59   MIP =    93.41
----------------------------------------------------------------------
```

In case the DICOPT run was not successful, or if one of the subproblems could not be solved, the listing file will contain all the status information provided by the solvers of the subproblems. Also for each iteration the configuration of the binary variables will be printed. This extra information can also be requested via the GAMS option:

```
option sysout = on ;
```

10 Special notes

This section covers some special topics of interest to users of DICOPT.

10.1 Stopping rule

Although the default stopping rule behaves quite well in practice, there some cases where it terminates too early. In this section we discuss the use of the stopping criteria.

When we run the example `procsel.gms` with stopping criterion 0, we see the following DICOPT log:

```
--- DICOPT: Starting major iteration 10
--- DICOPT: Search terminated: infeasible MIP master problem
--- DICOPT: Log File:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.06        8      0      conopt2
  MIP    1         2.48869      0.16        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt2
  MIP    2         2.17864      0.10       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt2
  MIP    3         1.42129      0.11       12      0      cplex
  NLP    4         1.41100      0.00        8      0      conopt2
  MIP    4         0.00000      0.22       23      0      cplex
  NLP    5         0.00000      0.00        3      0      conopt2
  MIP    5        -0.27778      0.16       22      0      cplex
  NLP    6        -0.27778      0.00        3      0      conopt2
  MIP    6        -1.00000      0.16       21      0      cplex
  NLP    7        -1.00000      0.00        3      0      conopt2
  MIP    7        -1.50000      0.22       16      0      cplex
  NLP    8        -1.50000      0.00        3      0      conopt2
  MIP    8        -2.50000      0.11       16      0      cplex
  NLP    9        -2.50000      0.00        3      0      conopt2
  MIP    9        *Infeas*      0.11        0      0      cplex
--- DICOPT: Terminating...
--- DICOPT: Stopped on infeasible MIP
```

```
      The search was stopped because the last MIP problem
      was infeasible. DICOPT will not be able to find
      a better integer solution.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSEL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion
```

This example shows some behavioral features that are not uncommon for other MINLP models. First, DICOPT finds often the best integer solution in the first few major iterations. Second, in many cases as soon as the NLP's start to give worse integer solution, no better integer solution will be found anymore. This observation is the motivation to make stopping option 2 where DICOPT stops as soon as the NLP's start to deteriorate the default stopping rule. In this example DICOPT would have stopped in major iteration 4 (you can verify this in the previous section). In many cases this will indeed give the best integer solution. For this problem, DICOPT has indeed found the global optimum.

Based on experience with other models we find that the default stopping rule (stop when the NLP becomes worse) performs well in practice. In many cases it finds the global optimum solution, for both convex and non-convex problems. In some cases however, it may provide a sub-optimal solution. In case you want more reassurance that no good integer solutions are missed you can use one of the other stopping rules.

Changing the MIP or NLP solver can change the path that DICOPT follows since the sub-problems may have non-unique solutions. The optimum stopping rule for a particular problem depends on the MIP and NLP solvers used.

In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut-off with `stop 1`. This option is however the best stopping criterion for convex problems.

10.2 Solving the NLP problems

In case the relaxed NLP and/or the other NLP sub-problems are very difficult, using a combination of NLP solvers has been found to be effective. For example, MINOS has much more difficulties to establish if a model is infeasible, so one would like to use CONOPT for NLP subproblems that are either infeasible or barely feasible. The `nlpsolver` option can be used to specify the NLP solver to be used for each iteration.

Infeasible NLP sub-problems can be problematic for DICOPT. Those sub-problems can not be used to form a new linearization. Effectively only the current integer configuration is excluded from further consideration by adding appropriate integer cuts, but otherwise an infeasible NLP sub-problem provides no useful information to be used by the DICOPT algorithm. If your model shows many infeasible NLP sub-problems you can try to add slack variables and add them with a penalty to the objective function.

Assume your model is of the form:

$$\begin{aligned} \min\ & f(x,y) \\ & g(x,y) \sim b \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{aligned} \tag{12}$$

where $\sim$ is a vector of relational operators $\{\leq, =, \geq\}$. x are continuous variables and y are the binary variables. If many of the NLP subproblems are infeasible, we can try the following "elastic" formulation:

$$\begin{aligned} \min\ & f(x,y) + M \sum_i (s_i^+ + s_i^-) \\ & y = y^B + s^+ - s^- \\ & g(x,y) \sim b \\ & \ell \leq x \leq u \\ & 0 \leq y \leq 1 \\ & 0 \leq s^+, s^- \leq 1 \\ & y^B \in \{0,1\} \end{aligned} \tag{13}$$

I.e. the variables y are relaxed to be continuous with bounds $[0,1]$, and binary variables y^B are introduced, that are related to the variables y through a set of the slack variables s^+, s^-. The slack variables are added to the objective with a penalty parameter M. The choice of a value for M depends on the size of $f(x,y)$, on the behavior of the model, etc. Typical values are 100, or 1000.

10.3 Solving the MIP master problems

When there are many discrete variables, the MIP master problems may become expensive to solve. One of the first thing to try is to see if a different MIP solver can solve your particular problems more efficiently.

Different formulations can have dramatic impact on the performance of MIP solvers. Therefore it is advised to try out several alternative formulations. The use of priorities can have a big impact on some models. It is possible to specify a nonzero value for `OPTCA` and `OPTCR` in order to prevent the MIP solver to spend an unacceptable long time in proving optimality of MIP master problems.

If the MIP master problem is infeasible, the DICOPT solver will terminate. In this case you may want to try the same reformulation as discussed in the previous paragraph.

10.4 Model debugging

In this paragraph we discuss a few techniques that can be helpful in debugging your MINLP model.

- Start with solving the model as an `RMINLP` model. Make sure this model solves reliably before solving it as a proper `MINLP` model. If you have access to different NLP solvers, make sure the `RMINLP` model solves smoothly with all NLP solvers. Especially CONOPT can generate useful diagnostics such as Jacobian elements (i.e. matrix elements) that become too large.
- Try different NLP and MIP solvers on the subproblems. Example: use the GAMS statement "`OPTION NLP=CONOPT3;`" to solve all NLP subproblem using the solver CONOPT version 3.
- The GAMS option statement "`OPTION SYSOUT = ON;`" can generate extra solver information that can be helpful to diagnose problems.
- If many of the NLP subproblems are infeasible, add slacks as described in section 10.2.
- Run DICOPT in pedantic mode by using the DICOPT option: "`CONTINUE 0`." Make sure all NLP subproblems solve to optimality.
- Don't allow any nonlinear function evaluation errors, i.e. keep the `DOMLIM` limit at zero. See the discussion on `DOMLIM` in section 7.1.
- If you have access to another MINLP solver such as SBB, try to use a different solver on your model. To select SBB use the following GAMS option statement: "`OPTION MINLP=SBB;`."
- Individual NLP or MIP subproblems can be extracted from the MINLP by using the `CONVERT` solver. It will write a model in scalar GAMS notation, which can then be solved using any GAMS NLP or MIP solver. E.g. to generate the second NLP subproblem, you can use the following DICOPT option: "`NLPSOLVER CONOPT2 CONVERT`." The model will be written to the file `GAMS.GMS`. A disadvantage of this technique is that some precision is lost due to the fact that files are being written in plain ASCII. The advantage is that you can visually inspect these files and look for possible problems such as poor scaling.

References

[1] M. A. Duran and I. E. Grossmann, *An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs*, Mathematical Programming, 36 (1986), pp. 307–339.

[2] E. Kalvelagen, *Model building with GAMS*, to appear.

[3] G. R. Kocis and I. E. Grossmann, *Relaxation Strategy for the Structural Optimization of Process Flowsheets*, Industrial and Engineering Chemistry Research, 26 (1987), pp. 1869–1880.

[4] G. R. Kocis and I. E. Grossmann, *Computational Experience with DICOPT solving MINLP Problems in Process Systems Engineering*, Computers and Chemical Engineering, 13 (1989), pp. 307–315.

[5] J. Viswanathan and I. E. Grossmann, *A combined Penalty Function and Outer Approximation Method for MINLP Optimization*, Computers and Chemical Engineering, 14 (1990), pp. 769–782.

[6] H. P. Williams, *Model Building in Mathematical Programming*, 4-th edition (1999), Wiley.

GAMBAS: A Program for Saving an Advanced Basis for GAMS

Version 1.0

by

Bruce McCarl
Professor
Texas A&M University
College Station, TX 77843-2124
McCarl@Tamu.edu

June 25,1996

GAMSBAS: A Program for Saving an Advanced Basis for GAMS

A program (GAMSBAS) has been written which saves information providing an advanced basis for a GAMS model. This information contains the shadow price, variable levels, and reduced costs, and is saved in a GAMS readable file. The file can, in turn, be included in subsequent GAMS models providing an advanced basis.

General Notes

Use of this procedure is relevant in cases where there are alterations in the data before a SOLVE statement in a large already solved model. The model should usually have saved restart files and the alterations should require rerunning the model from scratch. The general use of the procedure requires restarting GAMS after a solve and executing with a procedure which saves the basis information. The resultant data will be written on the file *.BAS, where "*" is the name of the GAMS input file. This basis then can be included in subsequent runs. An example of this procedure is given below.

The basis file should never be used for one time solution of a problem and rarely for solution of a file without use of restart files. One should only use this procedure with large models when one has to manipulate some of the original data sets equations, or variables before the first solve statement such that the model has to be restarted from scratch. One might also wish to preserve a basis from an alternative run.

The implementation of GAMSBAS causes it to go through several steps. When the procedure is first called GAMS generates the model and sends it out to the solver. In turn, GAMSBAS examines the problem and selects the solver to be used. Ordinarily, the default solver for the problem type (whether linear, nonlinear, or integer) is used. Users may exercise control over this process by using the options file (GAMSBAS.OPT) as described below. In turn, the solver is invoked and then GAMSBAS writes the basis.

During execution the program includes the line $OFFLISTING as the sixth line in the *.BAS file. This suppresses the listing of all but the first five lines of the basis in the file that includes it. Users wishing the full listing should delete this entry.

GAMS constructs a basis using information from the optimal solution. This ordinarily involves the level and marginal value of all variables plus an indicator of whether or not an equation has a shadow price. Degeneracies and alternative optimals complicate this process. GAMSBAS tries to overcome this by inserting EPS to indicate when a variable is basic or nonbasic.

Once the GAMSBAS information has been placed into GAMS the basis may not always be adequate. For example, a model which took over 100,000 iterations to get an initial solution

required 1200 iterations to reach optimality when restarted from its GAMSBAS basis. However, this reflected a considerable time saving.

Program Usage

There are three steps involved in using GAMSBAS. The first step involves changing the solver name in the GAMS file. This is done using the command:

```
OPTION LP=GAMSBAS
      or
OPTION NLP=GAMSBAS
      or
OPTION IP=GAMSBAS
```

The solver in this case is named GAMSBAS.

Second, restart the model and generate the basis file. Let's assume that the model name is BLOCKDIA. One would then execute the command GAMS BLOCKDIA with the solve option inserted before the solve command, as is done in Table 1, line 147. In turn, the file BLOCKDIA.BAS is generated. This file is listed in Table 2. Note, this file is just a set of GAMS replacement commands which inserts marginal values for the equations and marginal and level values for the variables (See the chapter on Basis formation in McCarl et al for an explanation of GAMS basis formation).

Third, an include command is entered right before the solve in the model to be restarted and the option selecting GAMSBAS as the solving program is normally eliminated. This is done in Table 3 in lines 147-8 (note the OPTION LP=GAMSBAS is commented out). Use of this procedure results in the model in Table 1 solving in 0 iterations after inclusion of the basis file as opposed to 23 iterations before inclusion of the basis file.

One may find that when a basis from one model is included in another model that the compiler may detect domain errors because the variables are defined over sets with different structures across the two models. One can suppress the domain errors by using the GAMS command $OFFUNI just before inclusion of the basis file and $ONUNI just after.

The OPTION FILE

GAMSBAS internally selects the solver to use. Users may override this choice by the use of the options file. There are keywords allowed in the options file. These are as follows

OPTION Name	Purpose
LP	Gives name of solver for LP problem
NLP	Gives name of solver for NLP problem
MIP	Gives name of solver for MIP problem
DNLP	Gives name of solver for DNLP problem
SOLVERNAME	Gives name of solver for problem to be used

In each case the option name is followed by the name of one of a licensed solvers. If the options file is empty then the default solvers will be used provided it matches the name of a solver GAMS knows about.

The GAMSBAS option file is called GAMSBAS.OPT. An example of a file could look like the following 2 lines

```
LP      OSL
MIP     LAMPS
```

One other important point regarding the option file involves the name of the active solver options file. As seen above the GAMSBAS.OPTION file does not include options commands such as those which should be submitted to MINOS for example. In all cases the program uses the default option filename for the particular solver. Thus if MINOS5 is being used the program looks for the solver option file on MINOS5.OPT.

References

Brooke, A., D. Kendrick, and A. Meeraus. GAMS: A User's Guide. The Scientific Press, South San Francisco, CA, 1988.

McCarl, B.A. “So Your GAMS Model Didn’t Work Right: A Guide to Model Repair.” Texas A&M University, College Station, TX, 1994.

McCarl, B.A., and T.H. Spreen. "Applied Mathematical Programming Using Algebraic Systems." Draft Book, Department of Agricultural Economics, Texas A&M University, College Station, TX, 1996.

Table 1. Example File

```
16  *     SECTION A       SET DEFINITION
17
18   SET  PRODUCT       TABLES CHAIRSSETS    /TABLES, CHAIRS, DINSETS/
19        TYPE          TYPES OF PRODUCT     /FUNCTIONAL ,FANCY/
20        RESOURCE      TYPES OF RESOURCES   /SMLLATHE,LRGLATHE,CARVER,LABOR,
                                                                      TOP/
21        METHOD        PRODUCTION METHODS   /NORMAL,MAXSML,MAXLRG/
22        PLANT         DIFFERENT PLANTS     /PLANT1, PLANT2, PLANT3/
23        SUBPRODUCT(PRODUCT)                /TABLES, CHAIRS/;
24
25  *     SECTION B       DATA DEFINITION
26
27   PARAMETER  SETCHAIR(TYPE)  CHAIRS CONTAINED IN EACH SET
28                            / FUNCTIONAL 4, FANCY 6/
29              TABLECOST(TYPE)  TABLECOST  /FUNCTIONAL 80,FANCY 100/;
30
31   TABLE  CHAIRCOST(METHOD,TYPE) CHAIR COST FOR DIFFERENT METHOD
32                                 FUNCTIONAL      FANCY
33              NORMAL                15            25
34              MAXSML                16            25.7
35              MAXLRG                16.5          26.6 ;
36
37   TABLE TB1(RESOURCE,TYPE,METHOD)   USE OF RESOURCES IN CHAIR PRODUCTION
38
39              FUNCTIONAL.NORMAL   FUNCTIONAL.MAXSML   FUNCTIONAL.MAXLRG
40  SMLLATHE          0.8               1.30               0.20
41  LRGLATHE          0.5               0.20               1.30
42  CARVER            0.4               0.40               0.40
43  LABOR             1.0               1.05               1.10
44   +
45
46               FANCY.NORMAL       FANCY.MAXSML        FANCY.MAXLRG
47  SMLLATHE         1.2               1.7                 0.50
48  LRGLATHE         0.7               0.30                1.50
49  CARVER           1.0               1.00                1.00
50  LABOR            0.8               0.82                0.84;
51
52   TABLE TB2(RESOURCE,TYPE)     USE OF RESOURCES IN TABLE PRODUCTION
53
54                  FUNCTIONAL     FANCY
55     LABOR          3             5
56     TOP            1             1;
57
58   TABLE TRANSCOST(SUBPRODUCT, PLANT,TYPE) TRANSPORT COST TO PLANT1
59
60               PLANT1.FUNCTIONAL     PLANT2.FUNCTIONAL  PLANT3.FUNCTIONAL
61      CHAIRS                                 5                7
62      TABLES                                                  20
63     +
64               PLANT1.FANCY     PLANT2.FANCY     PLANT3.FANCY
65      CHAIRS                         5                7
66      TABLES                                         20;
67
68   TABLE PRICE(PRODUCT,TYPE) PRICE OF CHAIRS
69                     FUNCTIONAL     FANCY
70          CHAIRS       82            105
71          TABLES      200            300
72          DINSETS     600           1100;
73
74   TABLE RESORAVAIL(RESOURCE,PLANT) RESOURCES AVAILABLE
75                   PLANT1      PLANT2      PLANT3
76       TOP          50                       40
77       SMLLATHE                 140         130
78       LRGLATHE                  90         100
79       CARVER                   120         110
80       LABOR        175         125         210;
```

Table 1. Example File (Continued)

```
 82   TABLE ACTIVITY(PRODUCT,PLANT) TELLS IF A PLANT SELLS A PRODUCT
 83               PLANT1   PLANT2   PLANT3
 84    TABLES      1                 1
 85    CHAIRS                1        1
 86    DINSETS     1                    ;
 87
 88  *   SECTION     C      MODEL  DEFINITION
 89
 90   POSITIVE VARIABLES
 91           MAKECHAIR(PLANT, TYPE,METHOD)  NUMBER OF CHAIRS MADE
 92           MAKETABLE(PLANT, TYPE)         NUMBER OF TABLES MADE
 93           TRNSPORT(PLANT,SUBPRODUCT,TYPE) NUMBER OF ITEMS TRANSPORTED
 94           SELL(PLANT,PRODUCT,TYPE)        NUMBER OF ITEMS SOLD;
 95
 96   VARIABLES
 97           NETINCOME    NET REVENUE (PROFIT);
 98   EQUATIONS
 99            OBJT         OBJECTIVE FUNCTION ( NET REVENUE )
100            RESOUREQ(PLANT,RESOURCE)
101            LINKTABLE(TYPE)  OVERALL FIRM TABLE LINKAGE CONSTRAINTS
102            LINKCHAIR(TYPE)  OVERALL FIRM CHAIR LINKAGE CONSTRAINTS
103            TRNCHAIREQ(PLANT,TYPE) CHAIRS BALANCE FOR A PLANT
104            TRNTABLEEQ(PLANT,TYPE) TABLES BALANCE FOR A PLANT;
105
106    OBJT..       NETINCOME =E=
107        SUM((TYPE, PRODUCT, PLANT)$ACTIVITY(PRODUCT,PLANT),
108                    PRICE(PRODUCT,TYPE) * SELL(PLANT,PRODUCT,TYPE))
109      - SUM((PLANT,TYPE)$ACTIVITY("TABLES",PLANT),
110                MAKETABLE(PLANT,TYPE)*TABLECOST(TYPE))
111      - SUM((PLANT,TYPE,METHOD)$ACTIVITY("CHAIRS",PLANT),
112                MAKECHAIR(PLANT,TYPE,METHOD)* CHAIRCOST(METHOD,TYPE))
113      - SUM((PLANT,TYPE,SUBPRODUCT)$TRANSCOST(SUBPRODUCT,PLANT,TYPE),
114              TRANSCOST(SUBPRODUCT,PLANT,TYPE) * TRNSPORT(PLANT,SUBPRODUCT,
115
116    RESOUREQ(PLANT,RESOURCE)..
117          SUM((TYPE,METHOD)$ACTIVITY("CHAIRS",PLANT), TB1(RESOURCE, TYPE,METHOD)
118               * MAKECHAIR(PLANT,TYPE,METHOD)) + SUM(TYPE$TB2(RESOURCE, TYPE),
119               TB2(RESOURCE,TYPE) * MAKETABLE(PLANT,TYPE))
120               =L= RESORAVAIL(RESOURCE,PLANT) ;
121
122    LINKTABLE(TYPE)..
123              SUM(PRODUCT$ACTIVITY(PRODUCT,"PLANT1"), SELL("PLANT1", PRODUCT,TYPE))
124              =L= MAKETABLE("PLANT1",TYPE) +
125                  SUM(PLANT$TRANSCOST("TABLES",PLANT,TYPE),
126                      TRNSPORT(PLANT,"TABLES",TYPE));
127
128    LINKCHAIR(TYPE)..
129               SELL("PLANT1","DINSETS",TYPE) * SETCHAIR(TYPE)
130               =L= SUM(PLANT$TRANSCOST("CHAIRS",PLANT,TYPE),
131                   TRNSPORT(PLANT,"CHAIRS",TYPE));
132
133    TRNCHAIREQ(PLANT,TYPE)..
134               (TRNSPORT(PLANT,"CHAIRS",TYPE) + SELL(PLANT,"CHAIRS",TYPE))
135                $TRANSCOST("CHAIRS",PLANT,TYPE)
136                =L= SUM(METHOD$ACTIVITY("CHAIRS",PLANT),
137                    MAKECHAIR(PLANT,TYPE,METHOD));
138
139    TRNTABLEEQ(PLANT,TYPE)..
140               (TRNSPORT(PLANT,"TABLES",TYPE) + SELL(PLANT,"TABLES",TYPE)
141                - MAKETABLE(PLANT,TYPE))$TRANSCOST("TABLES",PLANT,TYPE)
142                =L= 0 ;
143
144   MODEL Furn /ALL/;
145
146  * SECTION D    SOLVE THE PROBLEM
147   option lp=gamsbas
148   SOLVE Furn USING LP MAXIMIZING NETINCOME; 149
```

Table 2. Basis File

```
OBJT.m  =    1.00000000000    ;
RESOUREQ.m ("PLANT1","LABOR")  =    44.0000000000    ;
RESOUREQ.m ("PLANT2","SMLLATHE")  =    47.7696526508    ;
RESOUREQ.m ("PLANT2","LRGLATHE")  =    38.8299817185    ;
RESOUREQ.m ("PLANT2","LABOR")  =    19.3692870201    ;
$ OFFLISTING;
RESOUREQ.m ("PLANT3","SMLLATHE")  =    18.4975609756    ;
RESOUREQ.m ("PLANT3","LRGLATHE")  =    12.1853658537    ;
RESOUREQ.m ("PLANT3","CARVER")  =    35.2731707317    ;
RESOUREQ.m ("PLANT3","LABOR")  =    40.0000000000    ;
LINKTABLE.m ("FUNCTIONAL")  =    212.000000000    ;
LINKTABLE.m ("FANCY")  =    320.000000000    ;
LINKCHAIR.m ("FUNCTIONAL")  =    97.0000000000    ;
LINKCHAIR.m ("FANCY")  =    130.000000000    ;
TRNCHAIREQ.m ("PLANT2","FUNCTIONAL")  =    92.0000000000    ;
TRNCHAIREQ.m ("PLANT2","FANCY")  =    125.000000000    ;
TRNCHAIREQ.m ("PLANT3","FUNCTIONAL")  =    90.0000000000    ;
TRNCHAIREQ.m ("PLANT3","FANCY")  =    123.000000000    ;
TRNTABLEEQ.m ("PLANT3","FUNCTIONAL")  =    200.000000000    ;
TRNTABLEEQ.m ("PLANT3","FANCY")  =    300.000000000    ;
MAKECHAIR.l ("PLANT2","FUNCTIONAL","NORMAL")  =    62.2333942718    ;
MAKECHAIR.m ("PLANT2","FUNCTIONAL","MAXSML")  =   -14.2042961609    ;
MAKECHAIR.m ("PLANT2","FUNCTIONAL","MAXLRG")  =   -5.83912248629    ;
MAKECHAIR.l ("PLANT2","FANCY","NORMAL")  =    73.0195003047    ;
MAKECHAIR.m ("PLANT2","FANCY","MAXSML")  =   -9.44021937843    ;
MAKECHAIR.l ("PLANT2","FANCY","MAXLRG")  =    5.17976843388    ;
MAKECHAIR.l ("PLANT3","FUNCTIONAL","NORMAL")  =    35.3658536585    ;
MAKECHAIR.m ("PLANT3","FUNCTIONAL","MAXSML")  =   -8.59317073171    ;
MAKECHAIR.m ("PLANT3","FUNCTIONAL","MAXLRG")  =   -4.14975609756    ;
MAKECHAIR.l ("PLANT3","FANCY","NORMAL")  =    76.8292682927    ;
MAKECHAIR.m ("PLANT3","FANCY","MAXSML")  =   -5.87463414634    ;
MAKECHAIR.l ("PLANT3","FANCY","MAXLRG")  =    19.0243902439    ;
MAKETABLE.l ("PLANT1","FUNCTIONAL")  =    24.3998119826    ;
MAKETABLE.l ("PLANT1","FANCY")  =    20.3601128105    ;
MAKETABLE.m ("PLANT2","FUNCTIONAL")  =   -58.1078610603    ;
MAKETABLE.m ("PLANT2","FANCY")  =   -96.8464351005    ;
MAKETABLE.l ("PLANT3","FANCY")  =    19.4380487805    ;
MAKETABLE.m ("PLANT3", "FUNCTIONAL") = EPS;
TRNSPORT.l ("PLANT2","CHAIRS","FUNCTIONAL")  =    62.2333942718    ;
TRNSPORT.l ("PLANT2","CHAIRS","FANCY")  =    78.1992687386    ;
TRNSPORT.m ("PLANT3","TABLES","FUNCTIONAL")  =   -8.00000000000    ;
TRNSPORT.l ("PLANT3","TABLES","FANCY")  =    8.64870840207    ;
TRNSPORT.l ("PLANT3","CHAIRS","FUNCTIONAL")  =    35.3658536585    ;
TRNSPORT.l ("PLANT3","CHAIRS","FANCY")  =    95.8536585366    ;
SELL.m ("PLANT1","TABLES","FUNCTIONAL")  =   -12.0000000000    ;
SELL.m ("PLANT1","TABLES","FANCY")  =   -20.0000000000    ;
SELL.l ("PLANT1","DINSETS","FUNCTIONAL")  =    24.3998119826    ;
SELL.l ("PLANT1","DINSETS","FANCY")  =    29.0088212125    ;
SELL.m ("PLANT2","CHAIRS","FUNCTIONAL")  =  -10.00000000000    ;
SELL.m ("PLANT2","CHAIRS","FANCY")  =   -20.0000000000    ;
SELL.l ("PLANT3","TABLES","FANCY")  =    10.7893403784    ;
SELL.m ("PLANT3","CHAIRS","FUNCTIONAL")  =   -8.00000000000    ;
SELL.m ("PLANT3","CHAIRS","FANCY")  =   -18.0000000000    ;
NETINCOME.l  =    36206.8788960    ;
```

Table 3. Example with Basis File Included

```
*     SECTION A       SET DEFINITION

 SET  PRODUCT     TABLES CHAIRSSETS    /TABLES, CHAIRS, DINSETS/
      TYPE        TYPES OF PRODUCT     /FUNCTIONAL ,FANCY/
      RESOURCE    TYPES OF RESOURCES  /SMLLATHE,LRGLATHE,CARVER,LABOR, TOP/
      METHOD        PRODUCTION METHODS   /NORMAL,MAXSML,MAXLRG/
      PLANT         DIFFERENT PLANTS     /PLANT1, PLANT2, PLANT3/
      SUBPRODUCT(PRODUCT)                /TABLES, CHAIRS/;

*     SECTION B       DATA DEFINITION

 PARAMETER  SETCHAIR(TYPE)  CHAIRS CONTAINED IN EACH SET
                          / FUNCTIONAL 4, FANCY 6/
           TABLECOST(TYPE)  TABLECOST  /FUNCTIONAL 80,FANCY 100/;

 TABLE  CHAIRCOST(METHOD,TYPE) CHAIR COST FOR DIFFERENT METHOD
                               FUNCTIONAL      FANCY
            NORMAL                15            25
            MAXSML                16            25.7
            MAXLRG                16.5          26.6 ;

 TABLE TB1(RESOURCE,TYPE,METHOD)   USE OF RESOURCES IN CHAIR PRODUCTION

            FUNCTIONAL.NORMAL   FUNCTIONAL.MAXSML   FUNCTIONAL.MAXLRG
SMLLATHE          0.8               1.30               0.20
LRGLATHE          0.5               0.20               1.30
CARVER            0.4               0.40               0.40
LABOR             1.0               1.05               1.10
 +

             FANCY.NORMAL       FANCY.MAXSML        FANCY.MAXLRG
SMLLATHE         1.2               1.7                 0.50
LRGLATHE         0.7               0.30                1.50
CARVER           1.0               1.00                1.00
LABOR            0.8               0.82                0.84;

 TABLE TB2(RESOURCE,TYPE)     USE OF RESOURCES IN TABLE PRODUCTION

               FUNCTIONAL     FANCY
   LABOR          3            5
   TOP            1            1;

 TABLE TRANSCOST(SUBPRODUCT, PLANT,TYPE) TRANSPORT COST TO PLANT1
             PLANT1.FUNCTIONAL     PLANT2.FUNCTIONAL  PLANT3.FUNCTIONAL
    CHAIRS                                  5               7
    TABLES                                                  20
   +
             PLANT1.FANCY     PLANT2.FANCY     PLANT3.FANCY
    CHAIRS                         5               7
    TABLES                                        20;

 TABLE PRICE(PRODUCT,TYPE) PRICE OF CHAIRS
                  FUNCTIONAL     FANCY
        CHAIRS       82            105
        TABLES      200            300
        DINSETS     600           1100;

 TABLE RESORAVAIL(RESOURCE,PLANT) RESOURCES AVAILABLE
                PLANT1      PLANT2      PLANT3
    TOP          50                       40
    SMLLATHE                 140         130
    LRGLATHE                  90         100
    CARVER                   120         110
    LABOR        175         125         210;

 TABLE ACTIVITY(PRODUCT,PLANT) TELLS IF A PLANT SELLS A PRODUCT
            PLANT1   PLANT2   PLANT3
```

Table 3. Example with Basis File Included (Continued)

```
 84    TABLES      1                1
 85    CHAIRS               1        1
 86    DINSETS     1                   ;
 87
 88  *   SECTION     C      MODEL  DEFINITION
 89
 90   POSITIVE VARIABLES
 91           MAKECHAIR(PLANT, TYPE,METHOD)  NUMBER OF CHAIRS MADE
 92           MAKETABLE(PLANT, TYPE)         NUMBER OF TABLES MADE
 93           TRNSPORT(PLANT,SUBPRODUCT,TYPE) NUMBER OF ITEMS TRANSPORTED
 94           SELL(PLANT,PRODUCT,TYPE)        NUMBER OF ITEMS SOLD;
 95
 96   VARIABLES
 97           NETINCOME    NET REVENUE (PROFIT);
 98   EQUATIONS
 99            OBJT         OBJECTIVE FUNCTION ( NET REVENUE )
100            RESOUREQ(PLANT,RESOURCE)
101            LINKTABLE(TYPE)  OVERALL FIRM TABLE LINKAGE CONSTRAINTS
102            LINKCHAIR(TYPE)  OVERALL FIRM CHAIR LINKAGE CONSTRAINTS
103            TRNCHAIREQ(PLANT,TYPE) CHAIRS BALANCE FOR A PLANT
104            TRNTABLEEQ(PLANT,TYPE) TABLES BALANCE FOR A PLANT;
105
106    OBJT..       NETINCOME =E=
107        SUM((TYPE, PRODUCT, PLANT)$ACTIVITY(PRODUCT,PLANT),
108                    PRICE(PRODUCT,TYPE) * SELL(PLANT,PRODUCT,TYPE))
109      - SUM((PLANT,TYPE)$ACTIVITY("TABLES",PLANT),
110                MAKETABLE(PLANT,TYPE)*TABLECOST(TYPE))
111      - SUM((PLANT,TYPE,METHOD)$ACTIVITY("CHAIRS",PLANT),
112                MAKECHAIR(PLANT,TYPE,METHOD)* CHAIRCOST(METHOD,TYPE))
113      - SUM((PLANT,TYPE,SUBPRODUCT)$TRANSCOST(SUBPRODUCT,PLANT,TYPE),
114               TRANSCOST(SUBPRODUCT,PLANT,TYPE) * TRNSPORT(PLANT,SUBPRODUCT, TYPE));
115
116    RESOUREQ(PLANT,RESOURCE)..
117                SUM((TYPE,METHOD)$ACTIVITY("CHAIRS",PLANT), TB1(RESOURCE,TYPE,METHOD)
118                * MAKECHAIR(PLANT,TYPE,METHOD)) + SUM(TYPE$TB2(RESOURCE, TYPE),
119                TB2(RESOURCE,TYPE) * MAKETABLE(PLANT,TYPE))
120                =L= RESORAVAIL(RESOURCE,PLANT) ;
121
122    LINKTABLE(TYPE)..
123              SUM(PRODUCT$ACTIVITY(PRODUCT,"PLANT1"), SELL("PLANT1",PRODUCT,TYPE))
124              =L= MAKETABLE("PLANT1",TYPE) +
125                  SUM(PLANT$TRANSCOST("TABLES",PLANT,TYPE),
126                      TRNSPORT(PLANT,"TABLES",TYPE));
127
128    LINKCHAIR(TYPE)..
129                SELL("PLANT1","DINSETS",TYPE) * SETCHAIR(TYPE)
130                =L= SUM(PLANT$TRANSCOST("CHAIRS",PLANT,TYPE),
131                    TRNSPORT(PLANT,"CHAIRS",TYPE));
132
133    TRNCHAIREQ(PLANT,TYPE)..
134                (TRNSPORT(PLANT,"CHAIRS",TYPE) + SELL(PLANT,"CHAIRS",TYPE))
135                 $TRANSCOST("CHAIRS",PLANT,TYPE)
136                 =L= SUM(METHOD$ACTIVITY("CHAIRS",PLANT),
137                     MAKECHAIR(PLANT,TYPE,METHOD));
138
139    TRNTABLEEQ(PLANT,TYPE)..
140                (TRNSPORT(PLANT,"TABLES",TYPE) + SELL(PLANT,"TABLES",TYPE)
141                 - MAKETABLE(PLANT,TYPE))$TRANSCOST("TABLES",PLANT,TYPE)
142                 =L= 0 ;
143
144   MODEL Furn /ALL/;
145
146  * SECTION D    SOLVE THE PROBLEM
147  * option lp=gamsbas
148  $INCLUDE  "blockdia.bas"
149  SOLVE Furn USING LP MAXIMIZING NETINCOME;
```

Abridged GAMSCHK USER DOCUMENTATION

Version 1.1

A System for Examining the Structure and Solution
Properties of Linear Programming Problems
Solved using GAMS

by

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

(409) 845-7504 (fax)
mccarl@tamu.edu

June 25,1998

GAMSCHK USER DOCUMENTATION

LIST OF TABLES

GAMSCHK USER DOCUMENTATION

This document describes procedures designed to aid users who wish to examine empirical GAMS models for possible flaws. The conceptual basis for many of the routines herein is supplied in McCarl and Spreen, and McCarl et.al.

This package of routines is designed for use on any GAMS platform, but for now is implemented on the HP, PC, DEC Alpha, IBM RS6000 and SUN workstations. The function of the specific components of GAMSCHK are to:

(a) List coefficients for user selected equations and/or variables using the DISPLAYCR procedure.

(b) List the characteristics of selected groups of variables and/or equations using MATCHIT.

(c) List the characteristics of equation and variable blocks using BLOCKLIST.

(d) Examine a GAMS model to see whether any variables and equations contain specification errors using ANALYSIS.

(e) Generate schematics depicting the characteristics of coefficients by variable and equation blocks using BLOCKPIC.

(f) Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude using PICTURE.

(g) Reconstruct the reduced cost of variables and the activity within equations after a model solution using POSTOPT.

(h) Help resolving problems with unbounded or infeasible models using NONOPT and ADVISORY.

General Notes on Package Usage

GAMSCHK must replace a solver. This is done using a GAMS option statement of the form:

```
OPTION LP= GAMSCHK;
        or
OPTION NLP=GAMSCHK;
        or
OPTION MIP=GAMSCHK;
```

which replaces either the NLP, LP, or MIP solver with GAMSCHK.[1] In turn, the user will invoke the solver using the statement:

```
SOLVE MODELNAME USING LP MINIMIZING OBJNAME;
```

where MODELNAME is the name used in the GAMS MODEL statement; OBJNAME is the objective function name for the model; and the type of solver that GAMSCHK has replaced which must also be able to solve this type of problem (LP, NLP, or MIP) is identified.

The following are examples of GAMS sequences which can be added to the GAMS file:

```
OPTION NLP=GAMSCHK;
SOLVE TRANSPORT USING NLP MINIMIZING Z;
        or
OPTION LP=GAMSCHK;
SOLVE FEED USING LP MINIMIZING COST;
        or
OPTION MIP=GAMSCHK;
SOLVE RESOURCE USING MIP MAXIMIZING PROFIT;
```

Selecting a Procedure and Providing Input -- the *.GCK File

GAMSCHK requires that the user indicate which procedures are to be employed. This is specified through the use of the *.GCK file where the * refers to the filename from the GAMS execution instruction[2]. The general form of that file is:

```
FIRST PROCEDURE NAME
        ITEM SELECTION INPUT

SECOND PROCEDURE NAME
        ITEM SELECTION INPUT
```

Spaces and capitalization are ignored in this input. For example, a *.GCK file could look like

```
DISPLAYCR
```

---

[1] In all cases, users will be able to replace the LP solver. Replacement of the other solvers depends on the solver licenses owned by the user.

[2] Thus, if the GAMS instructions are in the file called MYMODEL, and GAMS is invoked using the DOS command GAMS MYMODEL, then the GCK file would be called MYMODEL.GCK. If GAMS instructions are on the filename with a period in it then the name up to the period will be used, i.e., the GCK file associated with MYMODEL.IT would be MYMODEL.GCK

```
variables
        SELL(*,*,FANCY)
        maketable
Invariables
        transport(plant2,*,fancy)
Equations
        objT
        notthere
inequations
        resourceq(plant1)
PICTURE
```

The first procedure name in this case is DISPLAYCR and the following 10 lines indicate the items to be selected. Then, we also request PICTURE. Selection entries are treated using several assumptions. In particular:

1) If the *.GCK file is empty then it is assumed that the BLOCKPIC procedure is selected.

2) Spaces maybe freely used in the GCK input file.

3) Upper, lower, or mixed case input is accepted.

4) GAMSCHK recognizes certain words. These words are listed in Appendix A and cannot be used as variable or equation names.

The *.GCK file: General Notes on Item Selection

Some of the procedures permit selection of variables, equations or functions. Specifically, the DISPLAYCR, PICTURE, POSTOPT, and MATCHIT procedures accept input identifying the variables and equations to be utilized. Also NONOPT accepts limited input controlling its function. General observations about the selection requests are

1) Variables can be chosen by entering the word *VARIABLE or VARIABLES* possibly with a modifier, followed by variable selection statements.

2) Variables can also be selected using the *INEQUATION or INEQUATIONS* syntax followed by names of equations. Use of this syntax results in selection of variables with coefficients in the named equations.

3) Equations are selected by entering the keyword, *EQUATION or EQUATIONS* possibly with a modifier, followed by equation selection statements.

4) Equations can also be selected using the *INVARIABLE or INVARIABLES* syntax followed by names of variables. Use of this syntax results in selection of equations in which the named variables have coefficients.

5) Certain item selection modifier keywords can be used depending on procedure. The *INTERSECT* keyword works with procedures *DISPLAYCR* and *POSTOPT*. The *INEQUATION* and *INVARIABLE* keywords work with procedures *DISPLAYCR, PICTURE* and *POSTOPT*. *LISTEQUATION* and *LISTVARIABLE* keywords work with the *MATCHIT* procedure. *INSOLUTION, NOTINSOLUTION, BINDING,* and *NOTBINDING* keywords work with *POSTOPT.* The keywords *VERBOSE* and *IDENTIFY* work with NONOPT.

6) If variable or equation names do not follow the keyword, then usually all variables or equations are assumed selected.

When variables or equations are to be selected after an item selection keyword, a number of input conventions apply. These conventions are:

1) If a variable or equation name is entered without any following parentheses, then all cases for that variable or equation are selected.

2) The selection entries identify specific elements from among the sets over which the variables and equations are defined. In specifying these elements one can use various wild card entries as discussed below or an element name. **Note GAMS set or subset names cannot be used. Set membership information is not available to the GAMSCHK routines.**

3) Wild cards can be used to select items. An "*" will select any item. For example, "B*" will select anything starting with a B. "A?B" will select anything beginning with A, ending with B with one intervening alpha numeric character.

4) When individual elements are specified, you need not enclose them in quotes (").

5) Quotes must be specified to include set item names with spaces, and special characters. In that case wild cards do not work and all input up to the next quote is simply copied.

6) When the selected item has more dimensions than specified, then all later dimensions are handled as if a wild card were specified. For example, when a variable X is defined with reference to 4 sets in the GAMS instructions, but only 3 parameters are specified in the GAMSCHK input, then the request is handled as if all elements of the 4th are desired.

7) When the selected item has less dimensions in GAMS than in the item

selection input, then all additional dimensions are ignored. Thus, when a variable X is defined with reference to 3 sets in GAMS, but 4 parameters are specified in the item selection file, then the 4th specification is ignored.

8) Multiple selection statements can appear on successive lines of the *.GCK file. Output is ordered according to the way items are found in the GAMS file which is determined by the ordering of variables, equations, and set elements in the original GAMS input.

9) Error messages will be generated when an entry cannot be matched to a GAMS element.

10) Examples include

X(*,CLEVELAND)	which indicates that X will be selected for any element of the first set where the element in the second set equals CLEVELAND
X(SEATTLE)	when X is two dimensional selects all cases where the first set element is SEATTLE
X(SEATTLE,CHICAGO,Z)	when X is two dimensional selects the case where the first set element equals SEATTLE, and the second element equals CHICAGO. The third is ignored.
X	all X's will be selected
X(S*, C.O, Z)	when X is three dimensional selects where all X's with first element starting with S, second element beginning with C and ending with O and third element Z will be selected.
*	all variables or equations will be selected
{empty selection set}	all variables or equations will be selected

Procedure Output

In all cases the output generated by the procedure will be written to the *.LST file associated with the GAMS call. Thus, if the file is called MODEL with the *.GCK file (MODEL.GCK), then all output will be on MODEL.LST.

Nonlinear Terms

GAMS models examined with GAMSCHK may involve nonlinear terms. In such cases, GAMSCHK uses the value of the nonlinear term sent forth from GAMS which is an accurate marginal, not total value. GAMS develops this value based on the current level value of the variable. This will either be: a) the starting point selected by GAMS, if the model has not been solved, or b) the current solution value, if the model has been solved. The most accurate portrayals of the coefficients will be generated after the model has been solved through a GAMS SOLVE command before invoking GAMSCHK. Some cases may require a solution and/or the specification of a good starting point before using GAMSCHK. Also, nonlinear terms potentially cause misleading coefficients as those values are local marginal, not global, values determined by the current levels of the variables. Nonlinear terms are marked with *** in the DISPLAYCR, POSTOPT, and NONOPT output.

Entering Comments in the *.GCK File

The *.GCK file has been programmed so that users can enter comments. These comments can take one of two forms. Comments that begin with a hash mark are copied to the output when the program runs. Comments which begin with a question mark are simply overlooked. Thus, one can temporarily comment GAMSCHK selection statements making them inactive by putting in question marks. If multiple procedures are being run or if some sort of output is decided to screen in the computer output then the hash marks can be entered.

Controlling Page Width in the *.GCK File

When running multiple procedures, in particular the pictures with other procedures, it is often desirable to have some procedures run with wide page widths, but the rest with a narrower page width. The GCK file provides the option to narrow the page width using a PW= command. In particular, what one can do is run GAMS with a large page width, i.e. run GAMS BLOCK pw=200, then insert in the GCK file instructions which narrow that page width for selected procedures. Users should note that the page width can never be made any wider than the default page width when running with GAMS. Information in excess of the page width will be ignored. Thus, if the model is run under the default status which has a page width of 75 characters then GAMSCHK will reduce the page width down to the maximum page width allowed. Consequently, the pw= command can only be used to narrow the page width from the default page width, not increase it.

Running Multiple Procedures

GAMSCHK can run multiple procedures during one job. This is done by simply stacking the sequence of the commands in the .GCK file.

Use of the Procedures

The following section describes the procedures available in GAMSCHK and their input requirements.

DISPLAYCR

Brief Purpose: DISPLAYCR displays all coefficients from the empirical model for a set of user selected equations and variables. All nonzero coefficients under each selected variable or in each selected equation are displayed with the associated variable or equation name and coefficient value. The selection entries may refer to all terms in equations /under variables or only those coefficients at the intersection of the selected variables and equations.

Usage Notes: This option mirrors the GAMS LIMCOL and LIMROW options, but allows the user to select the specific items to be displayed. Partial displays within a variable or equation are also allowed using *INTERSECT*. Use of *VARIABLE* and *EQUATION* keywords followed by selection statements allows one to select variables and equations. Use of the *INVARIABLE* command allows users to select the equations which are associated with a particular variable. For example, if one is having trouble with a particular variable and wants to look at competition in the equations in which it appears, then selecting the variable under the *INVARIABLE* command will display the complete contents of all the equations in which the selected variables have coefficients. Similarly, the *INEQUATION* command will display the complete contents of all variables which fall in a particular equation. Nonlinear terms are marked with ***.
When the keyword *INTERSECT* is found then only the coefficients at the intersection of the specified equations and variables are selected. Use of *INTERSECT* with the *INVARIABLE* syntax results in the named variables and the equations in which they fall being selected. Similarly, use of *INTERSECT* with the *INEQUATION* syntax results in selection of the named equations and the variables which fall in those equations.

Note that when GAMS internal scaling features are employed

the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File : The keyword DISPLAYCR is entered followed by optional lines of item selection input identifying the variables and equations to be displayed. This file can contain the keywords *VARIABLE*, *INVARIABLE*, *EQUATION*, and *INEQUATION*, with each followed by a specification of the items to be selected using the procedure input specification conventions that were described above. The keyword *INTERSECT* can also be used. Several special cases are relevant:

a) If none of the above keywords are found after DISPLAYCR and another procedure name does not follow, then the input is assumed to identify variables.

b) If input is found but the *VARIABLE* or *INEQUATION* keyword cannot be found then no variables are assumed selected.

c) If the *VARIABLE* keyword is entered, but is followed by the end of file or an Appendix A reserved word and *INEQUATION* does not appear, then all variables are assumed selected.

d) If the *EQUATION* or *INVARIABLE* keyword cannot be found, then no equations are assumed selected.

e) If the *EQUATION* keyword is entered, but is followed by the end of the file or a reserved word and the *INVARIABLE* command does not occur, then all equations are assumed selected.

f) The keyword *INVARIABLE* is allowed. It should be followed by variable selection statements. In turn, DISPLAYCR selects all equations which have nonzero entries under the *INVARIABLE* selections.

g) The keyword *INEQUATION* may be used. It should be followed by equation selection statements. In turn, DISPLAYCR selects all variables which have nonzero entries in the *INEQUATION* selections.

h) The keyword *INTERSECT* causes only coefficients at the intersection of the specified equations and variables to be displayed. This occurs for all specifications in this run of DISPLAYCR. One should use DISPLAYCR again if some intersecting and some non-intersecting displays are desired.

i) When *INTERSECT* appears along with *INVARIABLE*, the named variable is selected along with all the equations in which it falls. Similarly, when *INTERSECT* and *INEQUATION* appear then all the named equations and the variables appearing in them are selected.

MATCHIT

Brief Purpose: MATCHIT retrieves the names and characteristics of selected variables and equations. The characteristics reported tell whether the items are nonlinear as well as reporting scaling characteristics and counts of the coefficients. MATCHIT will summarize the items which match a request or list all the items individually.

Usage Notes: The input to MATCHIT can include the keywords *VARIABLE* and *EQUATION* along with those keywords with the prefix *LIST* attached. When the *LIST* prefix is not used, the procedure summarizes the characteristics of all items which match the item requests counting the number of matching items, the number of those items which are nonlinear, the total coefficients under or in those items, the number of positive, negative, and nonlinear coefficients that fall under or in those items. This does not list the names of the individual items which match. If the *LIST* prefix is used (entering *LISTVARIABLE* or *LISTEQUATION*) then the individual matching items are printed in the order in which they are encountered. For each matching item the information tells whether it is nonlinear, how many total coefficients it has, the count of positive, negative, and nonlinear coefficients falling under it, and the minimum and maximum absolute values of coefficients under it (excluding the objective function coefficient).
Note that when GAMS internal scaling features are employed then by default scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File : This file contains the keyword MATCHIT, followed by optional item selection input data. The optional input identifies the variables and equations to be displayed. This input can contain the keywords *VARIABLE* or *LISTVARIABLE* followed by a specification of the variables to be selected using the procedure input specification conventions that were described above. This can be followed by the keyword *EQUATION* or *LISTEQUATION* and the specified entries.

Several special cases are relevant:

a) If the procedure name is not followed by any selection input, then a count of all variables and equations appears.

b) If the input is found, but the input does not begin with *VARIABLE, EQUATION, LISTVARIABLE*, or *LISTEQUATION* keywords, then the input is assumed to contain variable names.

c) If the *VARIABLE* keyword is entered, but is not followed by variable selection statements, and *LISTVARIABLE* does not appear, then all variables are assumed selected.

d) If the *EQUATION* or *LISTEQUATION* keyword cannot be found, then no equations are assumed selected.

e) If the *EQUATION* keyword is entered, but is not followed by equation selection statements or a *LISTEQUATION* entry, then all equations are assumed selected.

f) The keyword *LISTVARIABLE* is allowed. It should be followed by variable selection statements. In turn, MATCHIT lists all variables which fall under the request.

g) The keyword *LISTEQUATION* may also be used. It should be followed by equation selection statements. In turn, MATCHIT lists all equations which fall under the request.

h) If none of the above keywords are found, the input is assumed to identify variables.

ANALYSIS

Brief Purpose: Analyzes the structure of all variables and equations. Information is given on errors involving obvious model misspecifications causing redundancy, zero variable values, infeasibility, unboundedness, or obvious constraint relaxations in linear programs. The checks are those identified in Tables 1, 2 and 3.

Usage Notes: The analysis tests given in Tables 1 and 2 are utilized to determine if individual variables or equations in the model possess obvious specification errors. One test, for example, considers whether or not in a maximization problem a variable appears which has a positive return in the objective function, but no coefficients in the constraints indicating an obviously unbounded model. Similarly, information is provided on whether certain equations can never be satisfied. For example, tests examine whether an equality equation appears with a negative right hand side and all positives on the left hand side. Also tests see whether the bounds on variables preclude equation satisfaction or make equations redundant (Table 3). In ANALYSIS these tests are applied to each and every variable and equation. The BLOCKPIC and BLOCKLIST routines utilize the tests on a block by block basis. Thus, the messages will be triggered only if every variable or equation in that block has the same problem. Also interactions between variables and equations are not checked so ANALYSIS only finds flaws contained in individual variables/equations.

Input File: The keyword ANALYSIS is all that is accepted.

BLOCKLIST

Brief Purpose : The BLOCKLIST procedure displays the number and characteristics of the items in each GAMS variable and equation block.

Usage Notes: The characteristic information gives:

1) The variable sign restriction or equation inequality type.

2) The number of variables or equations in this block;

3) The number of variables or equations with at least one nonlinear term in this block.

4) The number of positive coefficients under the variables or in the equations.

5) The number of negative coefficients under the variables or in the equations.

6) The number of nonlinear coefficients under the variables or in the equations.

7) The largest coefficient in absolute value in this block;

8) The smallest coefficient in absolute value in this block. Analysis tests are also performed as discussed under the ANALYSIS procedure.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: No input other than the procedure name is needed.

BLOCKPIC

Brief Purpose: Generates model schematics and scaling information. The schematics depict coefficient signs, total and average number of coefficients within each GAMS equation and variable block.

Usage Notes: These schematics are designed to aid users in identifying flaws in coefficient placement and sign. The summary information on problem scaling characteristics is designed to help users in scaling data. The scaling information is usually reported after any GAMS scaling (using the variablename.scale and equationname.scale features) but before solver scaling. (The user can change whether descaling is done - see the options file). Analysis tests are done using the procedures in Tables 1 and 2.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver

options file.

Input File : The keyword BLOCKPIC is all that is recognized.

PICTURE

Brief Purpose: Generates a schematic depicting the location, sign and magnitude of coefficients for selected variables and equations. Users can use this schematic to help identify flaws in coefficient placement, magnitude, or sign. Reports are also generated on the number of individual elements in the pictured portions of each variable and equation.

Usage Notes: This output can be quite large, so PICTURE should only be used for small models or model components.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: Optional input instructions may appear after the PICTURE keyword. This input selects the variables and equations to be included. Only coefficients at the intersection of the selected variables and equations are portrayed. The selected item in the .GCK file can contain the keywords *VARIABLE*, or *INVARIABLE* followed by a specification of the selected variables using the procedure input specification conventions above. This can be followed by the keywords *EQUATION* or *INEQUATION* and the specified entries. Several special cases are also relevant:

a) If the *VARIABLE* or *INEQUATION* keywords cannot be found, then all variables are assumed selected.

b) If the *EQUATION* or *INVARIABLE* keywords cannot be found, then all equations are assumed to selected.

c) If the none of the *VARIABLE*, *INVARIABLE*, *EQUATION*, or *INEQUATION* keywords are found, everything is pictured and all other input is ignored.

d) When the *INVARIABLE* keyword is used, then all equations in which those variables have coefficients are selected along with the named variables.

e) When the *INEQUATION* keyword is used, then all variables which have coefficients in the named equations are selected along with the named equations.

POSTOPT

Brief Purpose: Does post optimality computations. In that capacity POSTOPT either:

a) Reconstructs the reduced cost of variables after a GAMS model solution. Modelers can use this information to discover why certain variables are nonbasic or why certain shadow prices take on particular values, or

b) Reconstructs the usage and supply across an equation after a GAMS model solution. Modelers can use this information to discover why certain variables or slacks take on particular values, as well as to find out where items within equations are produced and/or used.

Usage Notes: POSTOPT uses essentially the same input conventions as does DISPLAYCR. Thus, the usage notes in that selection are also relevant here. In addition:

1) POSTOPT requires a solution has been obtained GAMSCHK will automatically cause a solver to be invoked unless suppressed by the options file;

2) Nonlinear terms may not be accurate in the row sums as their marginal value not their total value is used but GAMS will have adjusted the right-hand sides for their presence; and

3) Attention can be restricted to only certain types of variables or equations. Variables that are *INSOLUTION* (Nonzero or with Zero marginals), *NOTINSOLUTION* (zero with a nonzero marginal) can be requested, *BINDING* or *NONBINDING* equations can be focussed on.

Note that when GAMS internal scaling features are employed, the default option is that the unscaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File : An optional input file is read in, indicating the specific variables desired using the conventions explained under DISPLAYCR above. In addition:

1) One can enter *INSOLUTION* to restrict attention to variables which are nonzero or have zero marginals.

2) One can enter *NOTINSOLUTION* to restrict attention to zero variables.

3) The above entries restrict alteration in all *VARIABLE* or *INEQUATION* selection statements in a POSTOPT run.

4) One can enter *BINDING* to only consider equations with zero slack. Similarly, *NONBINDING* considers equations with nonzero slack.

5) The above equation specifications restrict all sections by all *EQUATION* or *INVARIABLES* items in a POSTOPT run.

ADVISORY

Brief Purpose: To identify variables which could be unbounded or equations and variable bounds which could cause a model to be infeasible.

Usage Notes: The ADVISORY procedure causes a presolution report on the set of all: a) variables which could be unbounded and/or b) equations and variable bounds which could cause infeasibility. The tests used are summarized in Table 3. This procedure identifies all variables which would need to be bounded as well as all constraints which need artificial variables if one wishes to diagnose problems in a model. The same output is also generated by NONOPT but the ADVISORY version does not require a solution.

Input file: Just the word ADVISORY

NONOPT

Brief Purpose: To help diagnose unbounded and infeasible models.

Usage Notes: The NONOPT procedure can be used in either an informative mode or with models which terminate as unbounded or infeasible. NONOPT will look through an optimal model reporting all variables which may be potentially unbounded or infeasible and all equations which may be infeasible using the checks explained under the ADVISORY section. Also in an unbounded model NONOPT can report the names of unbounded or infeasible variables or equations as well as either budgeting or row summing them. NONOPT runs after a solution and causes a solve to occur.

Input File: NONOPT may be followed by optional keywords *IDENTIFY* or *VERBOSE*. The *IDENTIFY* keyword causes GAMSCHK to report potential unbounded variables and/or infeasible equations. *VERBOSE* causes full budgets and row summing as done by the POSTOPT procedure on infeasible equations, and/or variables as well as unbounded variables and/or equations. Only the last encountered of the *VERBOSE* or *IDENTIFY* keywords will be obeyed. The details on these options are as follows:

1) If the *IDENTIFY* keyword is used, then the rules in Table 3 are applied to the model. Identify also anticipates that large upper bounds and/or artificial variables may be present. In an optimal condition all variable and equation levels that have exponents greater than the user supplied level filter in the options file (or 6 by default) are identified as items which could be involved with an unbounded model. Similarly, all variables or equations with marginals greater in exponent than the user supplied marginal exponent filter will be identified as items potentially involved with an infeasible model.

2) When the *VERBOSE* keyword is read then all variables and equations which are listed as nonoptimal or infeasible are treated using the budgeting and row summing aspects of POSTOPT.

3) When no keyword is found and the model solution is not

optimal then the nonoptimal equations, infeasible equations and/or nonoptimal variables are automatically listed.

Options File

GAMSCHK accepts an option file controlling solver choice (when needed); descaling; and size of the nonoptimal filters; the number of variable and column blocks selection entries allowed. The file is called GAMSCHK.OPT.

Solver Choice Options

GAMSCHK calls for the solution of the problem when the POSTOPT or NONOPT procedures are used. In doing this, GAMSCHK internally selects the default GAMS solver for a problem class. Users may override this choice using the solver options file. Users may also force or suppress the solution process.

There are 9 solver related keywords allowed in the options file. These are as follows:

OPTION	*Purpose*
LP	Gives name of solver for LP problems
NLP	Gives name of solver for NLP problems
MIP	Gives name of solver for MIP problems
DNLP	Gives name of solver for DNLP problems
SOLVERNAME	Gives name of solver to be used regardless of problem type
NOSOLVE	Suppresses solution of the problem
SOLVE	Forces solution of the problem
DESCALE	Controls treatment of scaling
OPTFILE	Solver options file number

In the first five cases, the option name is followed by the name of one of the licensed solvers. If the options file is empty, then the default solver will be used. If a solver name is given, then that solver will be used provided it matches the name of a solver GAMS recognizes.

When Should I Use *SOLVE* or *NOSOLVE*

Ordinarily GAMSCHK will cause a solver to be used if either the POSTOPT or the NONOPT options are used. However, users can force solutions under other cases or suppress solutions if desired.

One should only force a solution (using the *SOLVE* option) when one wishes to use the solution information after GAMSCHK is done either to examine the solution output or do post optimality calculations. Forcing a solution will not cause GAMSCHK to have improved

representations of nonlinear terms. That will only occur when a *SOLVE* statement is executed before the *SOLVE* statement involving GAMSCHK.

Control of Number of Variable and Row Selections Allowed

The GAMSCHK program uses an upper estimate on the number of variable or equation blocks. In rare circumstances users may wish to override this choice. The options for this are

OPTION	*Purpose*
VARBLOCK	Maximum number of variable blocks allowed
EQUBLOCK	Maximum number of equation blocks allowed

These options are followed by a number, but should not be routinely used.

Scaling

GAMS users may be utilizing internal features which involve scaling through the Modelname.SCALEOPT=1, VariableName.SCALE, and EquationName.SCALE options. GAMSCHK can work with these options to create output which reflects scaled, unscaled or partially unscaled output. In particular, the command DESCALE can be entered with one of three options: *NEVER, ALL,* or *PART*. If you enter *NEVER*, then none of the model output will be descaled. If you enter *ALL*, then all of the model output will be descaled. The third option is to use *PART*. In that case the NONOPT and POSTOPT output will be descaled whereas scaled information will be displayed for PICTURE, BLOCKPIC, BLOCKLIST, MATCHIT and DISPLAYCR. The *PART* option allows investigation of scaling. If you do not enter a *DESCALE* option then all information will be reported as if the *PART* option was chosen.

NONOPT Filters

The NONOPT model in "*IDENTIFY*" mode checks through a model solution to identify large marginals and/or large variable values. The limits on these checks are provided by two options

OPTION	*Purpose*
LEVELFILT	Numerical value of exponent on "unbounded levels"
MARGFILT	Numerical value of exponent on "infeasible marginals"

These options provide upper bounds on the exponents of the absolute values for the levels and marginals. They are followed by an integer which gives the exponent. Thus, entries like

LEVELFILT	7
MARGFILT	7

will cause the reporting of all marginals and levels which are greater in absolute value than 10^7

Example Options File

The GAMSCHK option file is called GAMSCHK.OPT. An example of a file could look like the following 6 lines

LP	OSL
MIP	LAMPS
VARBLOCK	50
SOLVE	
DESCALE	PART
LEVELFILT	4

Solver Options File

One other important aspect regarding the options file involves the use of a problem solver options file when a solver such as MINOS5, OSL, LAMPS etc. is also being used. As seen above the GAMSCHK.OPT does not recognize option commands such as those which would be submitted to the programming model solvers - OSL for example. In all cases GAMSCHK will cause the default option file for the solver to be used when invoking the solver. Thus if MINOS5 and the options file is invoked is being used, MINOS5 options are controlled by the option file MINO5.OPT while GAMSCHK.OPT controls GAMSCHK operation.

Users can change the number of the solver options file being used by using the OPTFILE parameter in the options file. OPTFILE 2 would cause use of solver options file .OP2.

Known Bugs

There are a few bugs that can cause GAMSCHK to report improper outputs or results. A list of the known bugs, their symptoms and a remedy is given below.

Symptom	Cause	Remedy

Zero Shadow Prices in POSTOPT	Old GAMS version or no Prior Solve	1) Make sure the model was solved, 2) if it was, do not suppress solve in option file, or 3) update to most recent GAMS version
Descaling Does Not Work	Old Version of GAMS	Update
GAMS Blows up after GAMSCHK Runs	Old GAMS version	Ignore, *.LST file, results are fine, can be fixed by updating to the most recent version of GAMS
POSTOPT has error in budgets equal to twice objective function coefficient for nonlinear maximizations	Old GAMS MINOS version	Switch to a minimization formulation or update GAMS/MINOS
ROWSUM does not fully account for the value of nonlinear terms in POSTOPT	Value of nonlinear terms sent from GAMS are only a marginal value	None planned. GAMSCHK would need reprogramming
Error message about size of VARBLOCK or EQNBLOCK	exceeded maximum number of blocks	Modify option file, enlarging or eliminating parameters
GAMSCHK won't run	Files are not properly installed	Recheck installation. If still doesn't work report to author
Zero shadow prices when using NOSOLVE	Old version of GAMS solvers or Shadow prices suppressed	Try changing GAMSCOMP.TXT lines 2 or 0 to 12 or 10, if that doesn't work update GAMS.

References

Brooke, A., D. Kendrick, and A. Meeraus. GAMS: A User's Guide. The Scientific Press, South San Francisco, CA, 1988.

McCarl, B.A. “So Your GAMS Model Didn’t Work Right: A Guide to Model Repair.” Texas A&M University, College Station, TX, 1994.

McCarl, B.A., and T.H. Spreen. "Applied Mathematical Programming Using Algebraic Systems." Draft Book, Department of Agricultural Economics, Texas A&M University, College Station, TX, 1996. On web page http://agrinet.tamu.edu/mccarl

Table 1. Conditions under which a modeler should be advised of potential difficulty for equations without nonlinear terms.

Type of Constraint	Count of coefficients under a variable of this type with a particular sign						Sign of RHS	Type of PS[a/]	Examples[b/]
	Nonnegative		Nonpositive		Unrestricted				
	+	-	+	-	+	-			
≤	≥ 0 [c/]	0	0	≥ 0	0	0	0	Zero Variables - Case 1	$\sum x \le 0$ [d/], $-\sum y \le 0$, $\sum x - \sum y \le 0$
	≥ 0	0	0	≥ 0	0	0	-	Infeasible -Case 2	$\sum x \le -k$, $-\sum y \le -k$, $\sum x - \sum y \le -k$
	0	≥ 0	≥ 0	0	0	0	+ or 0	Redundant -Case 3	$-\sum x \le +k$, $\sum y \le +k$, $-\sum x + \sum y \le k$
=	≥ 0	0	0	≥ 0	0	0	0	Zero Variables - Case 1	$\sum x = 0$, $-\sum y = 0$, $\sum x - \sum y = 0$
	0	≥ 0	≥ 0	0	0	0	0	Zero Variables - Case 1	$-\sum x = 0$, $\sum y = 0$, $-\sum x + \sum y = 0$
	≥ 0	0	0	≥ 0	0	0	-	Infeasible -Case 2	$\sum x - \sum y = -k$
	0	≥ 0	≥ 0	0	0	0	+	Infeasible -Case 2	$-\sum x + \sum y = k$
	0	0	0	0	≥ 0[e/]	≥ 0[e/]	0	Zero Variable - Case 1	$z=0$, $-z=0$
≥	0	≥ 0	≥ 0	0	0	0	0	Zero Variables - Case 1	$-\sum x \ge 0$, $\sum y \ge 0$, $-\sum x + \sum y \ge 0$
	0	≥ 0	≥ 0	0	0	0	0 or +	Infeasible -Case 2	$-\sum x \ge k$, $\sum y \ge k$, $-\sum x + \sum y \ge k$
	≥ 0	0	0	≥ 0	0	0	- or 0	Redundant -Case 3	$\sum x \ge -k$,$-\sum y \ge -k$, $\sum x - \sum y \ge -k$

a/ The PS cases indicate, because the variables in this equation follow this pattern, that:
1. The variables appearing with nonzeros in this equation are forced to equal zero.
2. This equation can never be satisfied and is obviously infeasible.
3. This equation is redundant. The nonnegativity conditions are a stronger restriction.

b/ In the examples x denotes indexed non-negative variables, y indexed non-positive variables, and z a single unrestricted variable.

c/ Here and in the cases below at least one nonzero must occur

d/ These entries give examples of the problem covered by each warning. Namely, in the first case examining only the nonnegative variables suppose all those variables have signs ≥ 0 but the right-hand-side is zero. Thus, we have $X \ge 0$ and $X \le 0$ which implies $X = 0$. A warning is generated in that case.

e/ Only one coefficient is allowed.

Table 2. Conditions under which a modeler should be warned about variables in a maximization problem.

Type of Variable	Objective function coefficient sign	Number of a_{ij}'s of a sign in						PS[a/]	Examples
		≥ rows		= rows		in ≤ rows			
		+	-	+	-	+	-		
Nonnegative	+	≥ 0	0	0	0	0	≥ 0	Unbounded Variable case 1	max x[b/] , x + DQ ≥ a -x + EQ≤ b
	-	0	≥ 0	0	0	≥ 0	0	Zero optimal solution case 2	max -x -x + DQ ≥ a x + EQ ≤ b
	0	≥ 0	0	0	0	0	≥ 0	Variable Relaxes constraint case 3	max ox x +DQ ≥ a -x + DQ ≤ b
	0	≥ 0	0	≥ 0[c]	≥ 0[c]	0	≥ 0	Variable Relaxes constraint case 4	max ox x + DQ ≥a x+ FQ = g -x + EQ ≤ b
Nonpositive	-	0	≥ 0	0	0	≥ 0	0	Unbounded Variable case 1	max -y -y + DQ ≥ a y + EQ ≤ b
	+	≥ 0	0	0	0	0	≥ 0	Zero optimal solution case 2	max y[b/] , y + DQ ≥ a -y + EQ≤ b
	0	0	≥ 0	0	0	≥ 0	0	Variable Relaxes constraint case 3	max 0x -y + DQ ≥ a y + EQ ≤ b
	0	0	≥ 0	≥ 0[c]	≥ 0[c]	≥ 0	0	Variable Relaxes constraint case 4	max ox -y + DQ ≥a y+FQ = g y + EQ ≤ b
Unrestricted	+/-	0	0	0	0	0	0	Unbounded Variable case 1	max ± z

a/ PS cases are: The variables which satisfy this condition are:

1) Unbounded as they contribute to the objective function while satisfying the constraints.
2) Obviously zero since they consume constraint resources and have a cost in the objective function.
3) Warning this variable relaxes all constraints in which it appears
4) Warning this variable relaxes all the equality constraints in which it appears in one direction

b/ Here x(y) has a positive objective term and can be increased without ever violating any constraints so x(y) is unbounded.

c/ Only one coefficient can be present in the equality rows

Table 3. Conditions When Model Elements Could be Unbounded or Infeasible

Conditions for Potential Unbounded Variables -- Presence of Bounds

Types of Variables	Sign of Objective in Max Problem	Upper	Lower
≥ 0[a]	+	None	---[c]
≤ 0	-	---	None
Unrestricted	+	None	---
Unrestricted	---	---	None

Conditions for Potential Infeasibility Caused by Bounds on Variables

	Existence of Bounds	
Types of Variables	Lower	Upper
$\geq$ [b]0	+	---
≤ 0	---	---
Unrestricted	+	---
Unrestricted	---	---

Conditions for Potential Infeasibility in Equations

Types of Equations	RHS
$\leq$[d]	-
$\geq$	+
=	+ or -

[a] If a non negative variable has a positive objective function coefficient without an upper bound, then the variable could be unbounded.

[b] If a nonnegative variable has a positive lower bound then it could cause infeasibility.

[c] Any reasonable value can exist for this item

[d] When a less than or equal equation is present it may not be able to be satisfied if it has a negative RHS.

Table 4. Conditions for Potential Infeasibility or Redundancy in Equations Based on Bounds on Variables

	TYPE OF CONSTRAINT		PS
	≤ b	≥ b	
SUM OF THE SMALLEST VALUE[a]	> b	---	INFEASIBLE
	---	> b	REDUNDANT
SUM OF THE LARGEST VALUE[b]	---	< b	INFEASIBILE
	< b	---	REDUNDANT

Note:

a. Suppose X_j is bounded as follows, LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum which is either >b or <b, then

S= will be the smallest value which could happen in that sum. If the constraint is ≤b, then if S>b, we know that this constraint will never be satisfied. In the constraint is ≥b, then if S>b, we know that this constraint will not limit any possible X value. Hence, it is redundant.

b. Suppose X_j is bounded as follows, LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum which is either >b or <b, then

L= will be the largest value which could happen in that sum. If the constraint is ≤b, if L<b, we know that this constraint will not limit any possible X value. Hence, it is redundant. In the constraint is ≥b, then if L<b, we know that this constraint will never be satisfied.

c.Thanks to Paul Preckel for bringing these tests to the authors' attention.

Appendix A: Reserved Names

VARIABLE
VARIABLES
EQUATION
EQUATIONS
INVARIABLE
INVARIABLES
INEQUATION
INEQUATIONS
LISTVARIABLE
LISTVARIABLES
LISTEQUATION
LISTEQUATIONS
POSTOPT
DISPLAYCR
PICTURE
BLOCKPIC
ANALYSIS
MATCHIT
BLOCKLIST
NONOPT
INSOLUTION
NOTINSOLUTION
NONINSOLUTON
VERBOSE
ADVISORY
BINDING
NONBINDING
NOTBINDING
INTERSECT
IDENTIFY
PW=

Appendix B: GAMSCHK One Page Summary

Invoking GAMSCHK **OPTION LP=GAMSCHK**

Keywords allowed in GCK file

Keyword	Allowed SubKEYWORDS	Brief Description
DISPLAYCR	VARIABLE* INVARIABLE* EQUATION* INEQUATION* INTERSECT++	Displays coefficients of selected variables and equations Indicates variable selections follow Indicates equations are wanted in which selected variables fall Indicates equation selections follow Indicates variables are wanted that fall in selected equations Show coefficients which appear at intersections of selected var/eqn
MATCHIT	VARIABLE* LISTVARIABLE* EQUATION* LISTEQUATION*	List variable and equation names and summarize characteristics Summarizes all variables matching selection statements Lists each variable matching a selection statement Summarizes all equations matching a selection statement Lists each equation matching a selection statement
ANALYSIS		Checks for obvious structural defects
BLOCKLIST		Summarizes characteristics of variable and equation blocks
BLOCKPIC		Generates block level schematics
PICTURE	VARIABLE* INVARIABLE* EQUATION* INEQUATION*	Generates tableau schematics Indicates variable selections follow Indicates equations are wanted in which selected variables fall Indicates equation selections follow Indicates variables are wanted that fall in selected equations
POSTOPT	VARIABLE* INVARIABLE* EQUATION* INEQUATION* INTERSECT++ NOTINSOLUTION++ INSOLUTION++ BINDING++ NONBINDING++	Reconstructs reduced cost and equation activity Indicates variable selections follow Indicates equations are wanted in which selected variables fall Indicates equation selections follow Indicates variables are wanted that fall in selected equations Show coefficients which appear at intersections of selected var/eqn Only nonzero vars or those with zero reduced cost Only zero vars will be selected Only eqns with zero slack will be computed Only eqns with nonzero slack will be computed
ADVISORY		List potential infeasible and unbounded items
NONOPT	IDENTIFY VERBOSE	Lists potential or actual nonoptimal items Same as ADVISORY but after solution Does POSTOPT computations on nonoptimals

Other Notes

1) Items marked above with an * are followed by item selection statements.
2) Items marked with ++ modifiy the types of variables, equations and coefficients selected.
3) In item selection an * is a wild card for multiple characters while a . is a wildcard for one character.
4) Spaces and capitalization don't matter in any of the input.
5) Options file controls scaling, solver choice, nonopt filters and maximum allowed selections.
6) Page width is controlled by a PW= keyword but cannot exceed GAMS page width.
7) Lines beginning with a ? or a # are treated as comments.

MILES: A Mixed Inequality and nonLinear Equation Solver

August, 1993
(Revised, March 1997)

Thomas F. Rutherford
Department of Economics
University of Colorado
Boulder, Colorado 80309-0256

rutherford@colorado.edu

Abstract

MILES is a solver for nonlinear complementarity problems and nonlinear systems of equations. This solver can be accessed indirectly through GAMS/MPSGE or GAMS/MCP. This paper documents the solution algorithm, user options, and program output. The purpose of the paper is to provide users of GAMS/MPSGE and GAMS/MCP an overview of how the MCP solver works so that they can use the program effectively.

1. **Introduction**

MILES is a Fortran program for solving nonlinear complementarity problems and nonlinear systems of equations. The solution procedure is a generalized Newton method with a backtracking line search. This code is based on an algorithm investigated by Mathiesen (1985) who proposed a modeling format and sequential method for solving economic equilibrium models. The method is closely related to algorithms proposed by Robinson (1975), Hogan (1977), Eaves (1978) and Josephy (1979). In this implementation, subproblems are solved as linear complementarity problems (LCPs), using an extension of Lemke's almost-complementary pivoting scheme in which upper and lower bounds are represented implicitly. The linear solver employs the basis factorization package LUSOL, developed by Gill et al. (1991).

The class of problems for which MILES may be applied are referred to as "generalized" or "mixed" complementarity problems, which is defined as follows:

$$\text{Given:} \quad F: R^n \to R^n, \quad \ell, u \in R^n$$

$$\text{Find:} \quad z, w, v \in R^n$$

such that

$$F(z) = w - v$$

$$\ell \le z \le u, \quad w \ge 0, \quad v \ge 0$$

$$w^T(z-\ell) = 0, \quad v^T(u-z) = 0$$

When $\ell=-\infty$ and $u=\infty$ MCP reduces to a nonlinear system of equations. When $\ell=0$ and $u=+\infty$, the MCP is a nonlinear complementarity problem. Finite dimensional variational inequalities are also MCP. MCP includes inequality-constrained linear, quadratic and nonlinear programs as special cases, although for these problems standard optimization methods may be preferred. MCP models which are not optimization problems encompass a large class of interesting mathematical programs. Specific examples of MCP formulations are not provided here. See Rutherford (1992a) for MCP formulations arising in economics. Other examples are provided by Harker and Pang (1990) and Dirkse (1993).

There are two ways in which a problem may be presented to MILES:

(i) MILES may be used to solve computable general equilibrium models generated by MPSGE as a GAMS subsystem. In the MPSGE language, a model-builder specifies classes of nonlinear functions using a specialized tabular input format embedded within a GAMS program. Using benchmark quantities and prices, MPSGE automatically calibrates function coefficients and generates nonlinear equations and Jacobian matrices. Large, complicated systems of nonlinear equations may be implemented and analyzed very easily using this interface to MILES. An introduction to general equilibrium modeling with GAMS/MPSGE is provided by Rutherford (1992a).

(ii) MILES may be accessed as a GAMS subsystem using

variables and equations written in standard GAMS algebra and the syntax for "mixed complementarity problems" (MCP). If more than one MCP solver is available[1], the statement "OPTION MCP=MILES;" tells GAMS to use MILES as the MCP solution system. When problems are presented to MILES using the MCP format, the user specifies nonlinear functions using GAMS matrix algebra and the GAMS compiler automatically generates the Jacobian functions. An introduction to the GAMS/MCP modeling format is provided by Rutherford (1992b).

The purpose of this document is to provide users of MILES with an overview of how the solver works so that they can use the program more effectively. Section 2 introduces the Newton algorithm. Section 3 describes the implementation of Lemke's algorithm which is used to solve linear subproblems. Section 4 defines switches and tolerances which may be specified using the options file. Section 5 interprets the run-time log file which is normally directed to the screen. Section 6 interprets the status file and the detailed iteration reports which may be generated. Section 7 lists and suggests remedies for abnormal termination conditions.

2. **The Newton Algorithm**

The iterative procedure applied within MILES to solve

[1] There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse, 1992).

nonlinear complementarity problems is closely related to the classical Newton algorithm for nonlinear equations. This first part of this section reviews the classical procedure. A thorough introduction to these ideas is provided by Dennis and Schnabel (1983). For a practical perspective, see Press et al. (1986).

Newton algorithms for nonlinear equations begin with a local (Taylor series) approximation of the system of nonlinear equations. For a point z in the neighborhood of $\bar{z}$, the system of nonlinear functions is linearized:

$$LF(z) = F(\bar{z}) + \nabla F(\bar{z})(z-\bar{z})$$

Solving the linear system $LF(z) = 0$ provides the Newton direction from $\bar{z}$ which given by $d = -\nabla F^{-1} F(\bar{z})$.

Newton iteration k begins at point z^k. First, the linear model formed at z^k is solved to determine the associated "Newton direction", d^k. Second, a line search in direction d^k determines the scalar steplength λ and the subsequent iterate: $z^{k+1} = z^k + \lambda d^k$. An Armijo or "back-tracking" line search initially considers $\lambda = 1$. If $\|F(z^{z}+\lambda d^{k})\| \leq \|F(z^{k})\|$, the step size λ is adopted, otherwise λ is multiplied by a positive factor α, $\alpha < 1$, and the convergence test is reapplied. This procedure is repeated until either an improvement results or $\lambda < \underline{\lambda}$. When $\underline{\lambda} = 0$, a positive

step is taken provided that:[2]

$$\frac{d}{d\lambda}\| F(z^k+\lambda d^k)\| < 0.$$

Convergence theory for this algorithm is quite well developed. See, for example, Ortega and Rheinbolt (1970) or Dennis and Schnabel (1983). The most attractive aspect of the Newton scheme with the backtracking line search is that in the neighborhood of a well-behaved fixed point, λ=1 is the optimal step length and the rate of convergence can be quadratic. If this method finds a solution, it does so very quickly.

The application of Newton methods to nonlinear complementarity problems involves a modification of the search direction. Here, *d* solves a *linear complementarity problem* (LCP) rather than a linear system of equations. For iteration *k*, *d* solves:

$$F(z^k)+\nabla F(z^k)d-w+v=0$$
$$\ell \le d+z^k \le u,\ w\ge 0,\ v\ge 0$$
$$w^T(d+z^k-\ell) = v^T(u-d-z^k) = 0$$

Conceptually, we are solving for *d*, but in practice MILES solves the linear problem in terms of the original variables $z = z^k + d$:

---

[2] α and $\underline{\lambda}$ correspond to user-specified tolerances DMPFAC and MINSTP, respectively.

$$F(z^k) - \nabla F(z^k) z_k + \nabla F(z^k)\, z = w - v$$

$$\ell \le z \le u, \quad w \ge 0, \quad v \ge 0$$

$$w^T(z-\ell) = 0, \quad v^T(u-z) = 0$$

After computing the solution z, MILES sets $d^k = z - z^k$.

The linear subproblem incorporates upper and lower bounds on any or all of the variables, assuring that the iterative sequence always remains within the bounds: ($\ell \le z^k \le u$). This can be helpful when, as is often the case, $F()$ is undefined for some $z \in \mathbf{R}^n$.

Convergence of the Newton algorithm applied to MCP hinges on three questions: (i) Does the linearized problem always have a solution? (ii) If the linearized problem has a solution, does Lemke's algorithm find it? and (iii) Is it possible to show that the computed direction d^k will provide an "improvement" in the solution? Only for a limited class of functions $F()$ can all three questions be answered in the affirmative. For a much larger class of functions, the algorithm converges in practice but convergence is not "provable".[3]

The answer to question (iii) depends on the choice of a norm by which an improvement is measured. The introduction of bounds and complementarity conditions makes the calculation of an error index is made more complicated. In MILES, the deviation associated with a candidate solution z, $\mathscr{E}(z)$, is based on a

[3] Kaneko (1978) provides some convergence theory for the linearized subproblem.

measure of the extent to which z, w and v violate applicable upper and lower bounds and complementarity conditions.

Evaluating Convergence

Let δ_i^L and δ_i^U be indicator variables for whether z_i is off its lower or upper bound. These are defined as:[4]

$$\delta_i^L = \min(1,(z_i-\ell_i)^+), \text{ and } \delta_i^U = \min(1,(u_i-z_i)^+).$$

Given z, MILES uses the value of $F(z)$ to implicitly define the slack variables w and v:

$$w_i = F_i(z)^+, \quad v_i = \left(-F_i(z)\right)^+.$$

There are two components to the error term associated with index i, one corresponding to z_i's violation of upper and lower bounds:

$$\varepsilon_i^B = (z_i-u_i)^+ + (\ell_i-z_i)^+,$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon_i^C = \delta_i^L w_i + \delta_i^U v_i.$$

[4] In the following $x^+ = \max(x, 0)$.

The error assigned to point z is then taken:

$$\varepsilon(z) = \| \varepsilon^B(z) + \varepsilon^C(z) \|_p$$

for a pre-specified value of p = 1, 2 or $+\infty$.[5]

3. **Lemke's Method with Implicit Bounds**

A mixed linear complementarity problem has the form:

$$\text{Given: } M \in R^{nxn}, \quad q, \ell, u \in R^n$$

$$\text{Find: } z, w, v \in R^n$$

such that

$$Mz + q = w - v$$

$$\ell \le z \le u, \quad w \ge 0, \quad v \ge 0$$

$$w^T(z - \ell) = 0, \quad v^T(u - z) = 0$$

In the Newton subproblem at iteration k, the LCP data are given by $q = F(z^k) - \nabla F(z^k)\, z^k$ and $M = \nabla F(z^k)$.

The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

$$B x^B + N x^N = q$$

where $x^B \in \mathbf{R}^n$, $x^N \in \mathbf{R}^{2n}$, and the tableau [B | N] is a conformal "complementary permutation" of [-M | I | -I]. That is, every

[5] Parameter p may be selected with input parameter NORM. The default value for p is $+\infty$.

column i in B must either be the ith column of M, I or $-I$, while the corresponding columns i and $i+n$ in N must be the two columns which were not selected for B.

To move from the problem defined in terms of z, w and v to the problem defined in terms of x^B and x^N, we assign upper and lower bounds for the x^B variables as follows:

$$\underline{x}_i^B = \begin{cases} \ell_i & \text{if } x_i^B = z_i \\ 0 & \text{if } x_i^B = w_i \text{ or } v_i \end{cases}$$

$$\overline{x}_i^B = \begin{cases} u_i & \text{if } x_i^B = z_i \\ \infty & \text{if } x_i^B = w_i \text{ or } v_i \end{cases}$$

The values of the non-basic variables x_i^N and x_{i+n}^N are determined by the assignment of x_i^B:

$$x_i^B = \begin{cases} z_i \Rightarrow \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = v_i = 0 \end{cases} \\ w_i \Rightarrow \begin{cases} x_i^N = z_i = \ell_i \\ x_{i+n}^N = v_i = 0 \end{cases} \\ v_i \Rightarrow \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = z_i = u_i \end{cases} \end{cases}$$

In words: if z_i is basic then both w_i and v_i equal zero. If z_i is non-basic at its lower bound, then w_i is possibly non-zero and v_i is non-basic at zero. If z_i is non-basic at its upper bound, then v_i is possibly non-zero and w_i is non-basic at zero.

Conceptually, we could solve the LCP by evaluating 3^n linear systems of the form:

$$x^B = B^{-1}\left(q - N\,x^N\right)$$

Lemke's pivoting algorithm provides a procedure for finding a solution by sequentially evaluating some (hopefully small) subset of these 3^n alternative linear systems.

Initialization

Let B^0 denote the initial basis matrix.[6] The initial values for basic variables are then:

$$\hat{x}^B = (B^0)^{-1}(q - N\,\hat{x}^N)$$

If $\underline{x}^B \le \hat{x}^B \le \overline{x}^B$, then the initial basis is feasible and the complementarity problem is solved.[7] Otherwise, MILES introduces

---

[6] In MILES, B^0 is chosen using the initially assigned values for z. When $z_i \le \ell_i$, then $x^B_i = w_i$; when $z_i \ge u_i$, then $x^B_i = v_i$; otherwise $x^B_i = z_i$.

[7] The present version of the code simply sets $B^0 = -I$ and $x^B = w$ when the user-specified basis is singular. A subsequent version of the code will incorporate the algorithm described by Anstreicher, Lee and Rutherford [1992] for coping with singularity.

an artificial variable z_0 and an artificial column h. Basic variables are then expressed as follows:

$$x^B = \hat{x}^B - \tilde{h}\, z_0$$

where $\tilde{h}$ is the "transformed artificial column" (the untransformed column is $h = B^0\, \tilde{h}$). The coefficients of $\tilde{h}$ are selected so that:

(i) The values of "feasible" basis variables are unaffected by z_0: ($\underline{x}_i^B \le x_i^B \le \overline{x}_i^B \Rightarrow \tilde{h}_i = 0$).

(ii) The "most infeasible" basic variable ($i=p$) is driven to its upper or lower bound when $z_0 = 1$:

$$\tilde{h}_p = \begin{cases} \hat{x}_p^B - \overline{x}_p^B & \text{if } \tilde{x}_p^B > \overline{x}_p^B \\ \hat{x}_p^B - \underline{x}_p^B & \text{if } \tilde{x}_p^B < \underline{x}_p^B \end{cases}$$

(iii) All other *infeasible* basic variables assume values between their upper and lower bounds when z_0 increases to 1:

$$x_i^B = \begin{cases} 1 + \underline{x}_i^B & \text{if } \underline{x}_i^B > -\infty,\ \overline{x}_i^B = +\infty \\ \dfrac{\overline{x}_i^B + \underline{x}_i^B}{2} & \textit{if } \underline{x}_i^B > -\infty,\ \overline{x}_i^B < +\infty \\ \overline{x}_i^B - 1 & \text{if } \underline{x}_i^B = -\infty,\ \overline{x}_i^B < +\infty \end{cases}$$

Table 1 Pivot Sequence Rules for Lemke's Algorithm with Implicit Bounds

Exiting Variable	Entering Variable	Change in Non-basic Values
(i) z_i at lower bound	w_i increases from 0	$x_i^N = z_i = \ell_i$
(ii) z_i at upper bound	v_i increases from 0	$x_{i+n}^N = z_i = u_i$
(iii) w_i at 0	z_i increases from ℓ_i	$x_i^N = x_{i+n}^N = 0$
(iv) v_i at 0	z_i decreases from u_i	$x_i^N = x_{i+n}^N = 0$

Pivoting Rules

When z_0 enters the basis, it assumes a value of unity, and at this point (baring degeneracy), the subsequent pivot sequence is entirely determined. The entering variable in one iteration is determined by the exiting basic variable in the previous iteration. For example, if z_i were in B^0 and introducing z_0 caused z_i to move onto its lower bound, then the subsequent iteration introduces w_i. Conversely, if w_i were in B^0 and z_0 caused w_i to fall to zero, the subsequent iteration increases z_i from ℓ_i. Finally, if v_i were in B^0 and z_0's introduction caused v_i to fall to zero, the subsequent iteration *decreases* z_i from u_i.

The full set of pivoting rules is displayed in Table 1. One difference between this algorithm and the original Lemke (type III) pivoting scheme (see Lemke (1965), Garcia and Zangwill

(1981), or Cottle and Pang (1992)) is that structural variables (z_i's) may enter or exit the basis at their upper bound values. The algorithm, therefore, must distinguish between pivots in which the entering variable increases from a lower bound from those in which the entering variable decreases from an upper bound.

Another difference with the "usual" Lemke pivot procedure is that an entering structural variable may move from one bound to another. When this occurs, the subsequent pivot introduces the corresponding slack variable. For example, if z_i is increased from ℓ_i to u_i without driving a basic variable infeasible, then z_i becomes non-basic at u_i, and the subsequent pivot introduces v_i. This type of pivot may be interpreted as z_i entering and exiting the basis in a single step.[8]

In theory it is convenient to ignore degeneracy, while in practice degeneracy is unavoidable. The present algorithm does not incorporate a lexicographic scheme to avoid cycling, but it does implement a ratio test procedure which assures that when there is more than one candidate, priority is given to the most stable pivot. The admissible set of pivots is determined on both an absolute pivot tolerance (ZTOLPV) and a relative pivot

---

[8] If all structural variables are subject to finite upper and lower bounds, then no z_i variables may be part of a homogeneous solution adjacent to a secondary ray. This does not imply, however, that secondary rays are impossible when all z_i variables are bounded, as a ray may then be comprised of w_i and v_i variables.

tolerance (ZTOLRP). No pivot with absolute value smaller than min(ZTOLPV, ZTOLRP ‖V‖) is considered, where ‖V‖ is the norm of the incoming column.

Termination on a Secondary Ray

Lemke's algorithm terminates normally when the introduction of a new variable drives z_0 to zero. This is the desired outcome, but it does not always happen. The algorithm may be interrupted prematurely when no basic variable "blocks" an incoming variable, a condition known as "termination on a secondary ray". In anticipation of such outcomes, MILES maintains a record of the value of z_0 for successive iterations, and it saves basis information associated with the smallest observed value, z_0^*. (In Lemke's algorithm, the pivot sequence is determined without regard for the effect on z_0, and the value of the artificial variable may follow an erratic (non-monotone) path from its initial value of one to the final value of zero.)

When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with z_0^* are reinstalled. This basis (augmented with one column from the non-basic triplet to replace z_0) serves as B^0, and the algorithm is restarted. In some cases this procedure permits the pivot sequence to continue smoothly to a solution, while in others cases may only lead to another secondary ray.

4. **The Options File**

MILES accepts the same format options file regardless of how the system is being accessed, through GAMS/MPSGE or GAMS/MCP. The options file is a standard text file which is normally named MILES.OPT.[9] The following is a typical options file:

```
BEGIN SPECS
              ITLIMT = 50
              CONTOL = 1.0E-8
              LUSIZE = 16
          END SPECS
```

All entries are of the form "<keyword> = <value>", where keywords have at most 6 characters. The following are recognized keywords which may appear in the options file, identified by keyword, type and default value, and grouped according to function:

Termination control

CONTOL *r* Default = 1.0E-6

is the convergence tolerance. Whenever an iterate is encountered for which $\mathcal{E}(z)$ < CONTOL, the algorithm terminates. This corresponds to the GAMS/MINOS parameter "Row tolerance".

ITLIMT *i* Default = 25

is an upper bound on the number of Newton iterations. This

[9] When invoking MILES from within GAMS it is possible to use one of several option file names. See the README documentation with GAMS 2.25 for details.

corresponds to the GAMS/MINOS parameter "Major iterations".

ITERLIM *i* Default = 1000

is an upper bound on the cumulative number of Lemke iterations. When MILES is invoked from within a GAMS program (either with MPSGE or GAMS/MCP), <model>.ITERLIM can be used to set this value. This corresponds to the GAMS/MINOS parameter "Iterations limit".

NORM *r* Default = 3

defines the vector norm to be used for evaluating $\in(z)$. Acceptable values are 1, 2 or 3 which correspond to $p = 1, 2$ and $+\infty$.

NRFMAX *i* Default = 3

establishes an upper bound on the number of factorizations in any LCP. This avoids wasting lots of CPU if a subproblem proves difficult to solve.

NRSMAX *i* Default = 1

sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.

Basis factorization control

FACTIM *r* Default = 120.0

indicates the maximum number of CPU seconds between recalculation of the basis factors.

INVFRQ *i* Default = 200

determines the maximum number of Lemke iterations between

recalculation of the basis factors. This corresponds to the GAMS/MINOS parameter "Factorization frequency".

ITCH *i* Default = 30

indicates the frequency with which the factorization is checked. The number refers to the number of basis replacements operations between refinements. This corresponds to the GAMS/MINOS parameter "Check frequency".

Pivot Selection

PLINFY *r* Default = 1.D20

is the value assigned for "plus infinity" ("+INF" in GAMS notation).

ZTOLPV *r* Default = 3.644E-11 (EPS**(2./3.))[10]

is the absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.

ZTOLRP *r* Default = 3.644E-11 (EPS**(2./3.))

is the relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.

ZTOLZE *r* Default = 1.E-6

is used in the subproblem solution to determine when any

[10] A number of tolerances are set on the basis of the machine precision, EPS. For Lahey Fortran running on 80486 processor, EPS = 2.2D-16.

variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter "Feasibility tolerance".

Linearization and Data Control

SCALE *l* Default = .TRUE.

invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch "scale all variables".

ZTOLDA *r* Default = 1.483E-08 (EPS**(1/2))

sets a lower bound on the magnitude of nonzeros recognized in the linearization. All coefficients with absolute value less than ZTOLDA are ignored.

Newton Iteration Control

DMPFAC *r* Default = 0.5

is the Newton damping factor, represented by α above.

MINSTP *r* Default = 0.03

is the minimum Newton step length, represented by $\underline{\lambda}$ above.

Output Control

INVLOG *l* Default = .TRUE.

is a switch which requests LUSOL to generate a report with basis statistics following each refactorization.

LCPDMP *l* Default =.FALSE.

is a switch to generate a printout of the LCP data after

scaling.

LCPECH *l* Default = .FALSE.

is a switch to generate a printout of the LCP data before scaling, as evaluated.

LEVOUT *i* Default = 0

sets the level of debug output written to the log and status files. The lowest meaningful value is -1 and the highest is 3. This corresponds, roughly, to the GAMS/MINOS parameter "Print level"

PIVLOG *l* Default =.FALSE.

is a switch to generate a status file listing of the Lemke pivot sequence.

LUSOL parameters (as with MINOS 5.4, except LUSIZE)

LUSIZE *i* Default = 10

is used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix.

LPRINT *i* Default=0

is the print level, < 0 suppresses output.

= 0 gives error messages.

= 1 gives debug output from some of the other routines in LUSOL.

>= 2 gives the pivot row and column and the no. of rows and columns involved at each

elimination step in lu1fac.

MAXCOL *i* Default=5

in lu1fac is the maximum number of columns searched allowed in a Markowitz-type search for the next pivot element. For some of the factorization, the number of rows searched is maxrow = maxcol - 1.

ELMAX1 *r* Default=10.0

is the maximum multiplier allowed in L during factor.

ELMAX2 *r* Default=10

is the maximum multiplier allowed in L during updates.

SMALL *r* Default=EPS**(4./5.)

is the absolute tolerance for treating reals as zero. IBM double: 3.0d-13

UTOL1 *r* Default=EPS**(2./3.)

is the absolute tol for flagging small diagonals of U. IBM double: 3.7d-11

UTOL2 *r* Default=EPS**(2./3.)

is the relative tol for flagging small diagonals of U. IBM double: 3.7d-11

USPACE *r* Default=3.0

is a factor which limits waste space in U. In lu1fac, the row or column lists are compressed if their length exceeds uspace times the length of either file after the last compression.

DENS1 *r* Default=0.3

is the density at which the Markowitz strategy should search maxcol columns and no rows.

DENS2 *r* Default=0.6

is the density at which the Markowitz strategy should search only 1 column or (preferably) use a dense LU for all the remaining rows and columns.

5. **Log File Output**

The log file is intended for display on the screen in order to permit monitoring progress. Relatively little output is generated.

A sample iteration log is displayed in Table 2. This output is from two cases solved in succession. This and subsequent output comes from program TRNSP.FOR which calls the MILES library directly. (When MILES is invoked from within GAMS, at most one case is processed at a time.)

Table 2 Sample Iteration Log

```
MILES  (July 1993)                                        Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email:  TOM@GAMS.COM


Work space allocated           --    0.01 Mb

Initial deviation ........ 3.250E+02    P_01
Convergence tolerance .... 1.000E-06

   0    3.25E+02    1.00E+00 (P_01    )
   1    1.14E-13    1.00E+00 (W_02    )

Major iterations ........     1
Lemke pivots ............    10
Refactorizations ........     2
Deviation ............... 1.137E-13

Solved.


Work space allocated           --    0.01 Mb

Initial deviation ........ 5.750E+02    W_02
Convergence tolerance .... 1.000E-06

   0    5.75E+02    1.00E+00 (W_02    )
   1    2.51E+01    1.00E+00 (P_01    )
   2    4.53E+00    1.00E+00 (P_01    )
   3    1.16E+00    1.00E+00 (P_01    )
   4    3.05E-01    1.00E+00 (P_01    )
   5    8.08E-02    1.00E+00 (P_01    )
   6    2.14E-02    1.00E+00 (P_01    )
   7    5.68E-03    1.00E+00 (P_01    )
   8    1.51E-03    1.00E+00 (P_01    )
   9    4.00E-04    1.00E+00 (P_01    )
  10    1.06E-04    1.00E+00 (P_01    )
  11    2.82E-05    1.00E+00 (P_01    )
  12    7.47E-06    1.00E+00 (P_01    )
  13    1.98E-06    1.00E+00 (P_01    )
  14    5.26E-07    1.00E+00 (P_01    )

Major iterations ........    14
Lemke pivots ............    23
Refactorizations ........    15
Deviation ............... 5.262E-07

Solved.
```

The first line of the log output gives the MILES program date and version information. This information is important for bug reports.

The line beginning "Work space..." reports the amount of memory which has been allocated to solve the model - 10K for this example. Thereafter is reported the initial deviation together with the name of the variable associated with the largest imbalance ($\mathfrak{e}_i^B + \mathfrak{e}_i^C$). The next line reports the convergence tolerance.

The lines beginning 0 and 1 are the major iteration reports for those iterations. the number following the iteration number is the current deviation, and the third number is the Armijo step length. The name of the variable complementary to the equation with the largest associated deviation is reported in parenthesis at the end of the line.

Following the final iteration is a summary of iterations, refactorizations, amd final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

6. **Status File Output**

The status file reports more details regarding the solution process than are provided in the log file. Typically, this file is written to disk and examined only if a problem arises. Within

GAMS, the status file appears in the listing only following the GAMS statement "OPTION SYSOUT=ON;".

The level of output to the status file is determined by the options passed to the solver. In the default configuration, the status file receives all information written to the log file together a detailed listing of all switches and tolerances and a report of basis factorization statistics.

When output levels are increased from their default values using the options file, the status file can receive considerably more output to assist in debugging. Tables 3-6 present a status file generated with LEVOUT=3 (maximum), PIVLOG=T, and LCPECH=T.

The status file begins with the same header as the log file. Thereafter is a complete echo-print of the user-supplied option file when one is provided. Following the core allocation report is a full echo-print of control parameters, switches and tolerance as specified for the current run.

Table 4 contiues the status file. The iteration-by-iteration report of variable and function values is produced whenever LEVOUT $\geq$ 2. Table 4 also contains an LCP echo-print. This report has two sections: $ROWS and $COLUMNS. The four columns of numbers in the $ROWS section are the constant vector (q), the current estimate of level values for the associated variables (z), and the lower and upper bounds vectors (ℓ and u). The letters L and U which appear between the ROW and Z columns are used to identify variables which are non-basic at their lower

and upper bounds, respectively. In this example, all upper bounds equal $+\infty$, so no variables are non-basic at their upper bound.

By convention, only variable (and not equation names) appear in the status file. An equation is identified by the corresponding variable. We therefore see in the $COLUMNS: section of the matrix echo-print, the row names correspond to the names of z variables. The names assigned to variables z_i, w_i and v_i are z-<name i>, w-<name i> and v-<name i>, as shown in the $COLUMNS section. The nonzeros for w-<> and v-<> variables are not shown because they are assumed to be -/+I.

The status file output continues on Table 5 where the first half of the table reports output from the matrix scaling procedure, and the second half reports the messages associated with initiation of Lemke's procedure.

The "lu6chk warning" is a LUSOL report. Thereafter are two factorization reports. Two factorizations are undertaken here because the first basis was singular, so the program install all the lower bound slacks in place of the matrix defined by the initial values, z.

Following the second factorization report, at the bottom of Table 5 is a summary of initial pivot. "Infeasible in 3 rows." indicates that $\tilde{h}$ contains 3 nonzero elements. "Maximum infeasibility" reports the largest amount by which a structural variable violates an upper or lower bound. "Artificial column

with 3 elements." indicates that the vector $h = B^0 \tilde{h}$ contains 3 elements (note that in this case $B^0 = -I$ because the initial basis was singular, hence the equivalence between the number of nonzeros in $\tilde{h}$ and h.).

Table 6 displays the final section of the status file. At the top of page 6 is the Lemke iteration log. The columns are interpreted as follows:

ITER is the iteration index beginning with 0,

STATUS is a statistic representing the efficiency of the Lemke path. Formally, status is the ratio of the minimum number of pivots from B^0 to the current basis divided by the actual number of pivots. When the status is 1, Lemke's algorithm is performing virtually as efficiently as a direct factorization (apart from the overhead of basis factor updates.)

Z% indicates the percentage of columns in the basis are "structural" (z's).

Z0 indicates the value of the artificial variable. Notice that in this example, the artificial variable declines monotonically from its initial value of unity.

ERROR is a column in which the factorization error is reported, when it is computed. For this run,

ITCH=30 and hence no factorization errors are computed.

INFEAS. is a column in which the magnitude of the infeasibility introduced by the artificial column (defined using the box-norm) is reported. (In MILES the cover vector h contains many different nonzero values, not just 1's; so there may be a large difference between the magnitude of the artificial variable and the magnitude of the induced infeasibility.

PIVOTS reports the pivot magnitude in both absolute terms (the first number) and relative terms (the second number). The relative pivot size is the ratio of the pivot element to the norm of the incoming column.

IN/OUT report the indices (not names) of the incoming and outgoing columns for every iteration. Notice that Lemke's iteration log concludes with variable z_0 exiting the basis.

The convergence report for iteration 1 is no different from the report written to the log file, and following this is a second report of variable and function values. We see here that a solution has been obtained following a single subproblem. This is because the underlying problem is, in fact, linear.

The status file (for this case) concludes with an iteration summary identical to the log file report and a summary of how much CPU time was employed overall and within various subtasks. (Don’t be alarmed if the sum of the last five numbers does not add up to the first number - some cycles are not monitored precisely.)

Table 3 Status File with Debugging Output (page 1 of 4)

```
MILES  (July 1993)                                        Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email:  TOM@GAMS.COM



User supplied option file:

>BEGIN
>     PIVLOG = .TRUE.
>     LCPECH = .TRUE.
>     LEVOUT = 3
>END


Work space allocated           --    0.01 Mb



NEWTON algorithm control parameters:
    Major iteration limit ..(ITLIMT).          25
    Damping factor .........(DMPFAC).   5.000E-01
    Minimum step length ....(MINSTP).   1.000E-02
    Norm for deviation .....(NORM)...           3
    Convergence tolerance ..(CONTOL).   1.000E-06

LEMKE algorithm control parameters:
    Iteration limit .......(ITERLIM).        1000
    Factorization frequency (INVFRQ).         200
    Feasibility tolerance ..(ZTOLZE).   1.000E-06
    Coefficient tolerance ..(ZTOLDA).   1.483E-08
    Abs. pivot tolerance ...(ZTOLPV).   3.644E-11
    Rel. pivot tolerance ...(ZTOLRP).   3.644E-11
    Cover vector tolerance .(ZTOLZ0).   1.000E-06
    Scale every iteration ...(SCALE).   T
    Restart limit ..........(NRSMAX).           1

Output control switches:
    LCP echo print ...........(LCPECH).   F
    LCP dump .................(LCPDMP).   T
    Lemke inversion log ......(INVLOG).   T
    Lemke pivot log ......... (PIVLOG).   T

Initial deviation ........ 3.250E+02    P_01
Convergence tolerance .... 1.000E-06

================================
Convergence Report, Iteration  0
================================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00 (P_01    )
================================================================
```

Table 4 Status File with Debugging Output (page 2 of 4)

```
 Iteration   0 values.

                ROW             Z                       F
               --------    ------------            ------------
               X_01_01  L   0.00000E+00            -7.75000E-01
               X_01_02  L   0.00000E+00            -8.47000E-01
               X_01_03  L   0.00000E+00            -8.38000E-01
               X_02_01  L   0.00000E+00            -7.75000E-01
               X_02_02  L   0.00000E+00            -8.38000E-01
               X_02_03  L   0.00000E+00            -8.74000E-01
               W_01     L   0.00000E+00             3.25000E+02
               W_02     L   0.00000E+00             5.75000E+02
               P_01         1.00000E+00            -3.25000E+02
               P_02         1.00000E+00            -3.00000E+02
               P_03         1.00000E+00            -2.75000E+02

 ==================================
 Function Evaluation, Iteration:  0
 ==================================


$ROWS:
X_01_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_02    -1.53000004E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_03    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_02    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_03    -1.25999998E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
W_01       -3.25000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
W_02       -5.75000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
P_01        3.25000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_02        3.00000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_03        2.75000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
     ...    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00


$COLUMNS:
Z-X_01_01   W_01     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_01_02   W_01     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_01_03   W_01     -1.00000000E+00
            P_03      1.00000000E+00
Z-X_02_01   W_02     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_02_02   W_02     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_02_03   W_02     -1.00000000E+00
            P_03      1.00000000E+00
Z-W_01      X_01_01   1.00000000E+00
            X_01_02   1.00000000E+00
            X_01_03   1.00000000E+00
Z-W_02      X_02_01   1.00000000E+00
            X_02_02   1.00000000E+00
            X_02_03   1.00000000E+00
Z-P_01      X_01_01  -1.00000000E+00
            X_02_01  -1.00000000E+00
Z-P_02      X_01_02  -1.00000000E+00
            X_02_02  -1.00000000E+00
Z-P_03      X_01_03  -1.00000000E+00
            X_02_03  -1.00000000E+00
       ...       ...  0.00000000E+00
```

Table 5 Status File with Debugging Output (page 3 of 4)

```
SCALING LCP DATA
-------
            MIN ELEM    MAX ELEM       MAX COL RATIO

AFTER  0    1.00E+00    1.00E+00               1.00
AFTER  1    1.00E+00    1.00E+00               1.00
AFTER  2    1.00E+00    1.00E+00               1.00
AFTER  3    1.00E+00    1.00E+00               1.00


SCALING RESULTS:

      A(I,J) <= A(I,J) * R(I) / C(J)

  ROW         ROW    Z COLUMN    W COLUMN    V COLUMN
    1      1.0000      1.0000      1.0000      1.0000
    2      1.0000      1.0000      1.0000      1.0000
    3      1.0000      1.0000      1.0000      1.0000
    4      1.0000      1.0000      1.0000      1.0000
    5      1.0000      1.0000      1.0000      1.0000
    6      1.0000      1.0000      1.0000      1.0000
    7      1.0000      1.0000      1.0000      1.0000
    8      1.0000      1.0000      1.0000      1.0000
    9      1.0000      1.0000      1.0000      1.0000
   10      1.0000      1.0000      1.0000      1.0000
   11      1.0000      1.0000      1.0000      1.0000


lu6chk  warning.  The matrix appears to be singular.
    nrank =      8        rank of U
n - nrank =      3        rank deficiency
    nsing =      3        singularities
    jsing =     10        last singular column
    dumax =   1.00E+00    largest  triangular diag
    dumin =   1.00E+00    smallest triangular diag


LUSOL 5.4  FACTORIZATION STATISTICS
Compressns    0    Merit      0.00    LenL          0    LenU         14
Increase   0.00    M            11    UT           11    D1            0
Lmax    0.0E+00    Bmax    1.0E+00    Umax    1.0E+00    Umin    1.0E+00
Growth  1.0E+00    LT              0    BP            0    D2            0


LUSOL 5.4  FACTORIZATION STATISTICS
Compressns    0    Merit      0.00    LenL          0    LenU         11
Increase   0.00    M            11    UT           11    D1            0
Lmax    0.0E+00    Bmax    1.0E+00    Umax    1.0E+00    Umin    1.0E+00
Growth  1.0E+00    LT              0    BP            0    D2            0


CONSTRUCTING ARTIFICIAL COLUMN

--- Infeasible in   3 rows.
--- Maximum infeasibility:    3.250E+02
--- Artificial column with   3 elements.
--- Pivoting in row:   9 to replace column  20
--- Pivot element:   -3.250E+02
```

Table 6 Status File with Debugging Output (page 4 of 4)

```
                        LEMKE PIVOT STEPS
                        =================
ITER  STATUS    Z%   Z0     ERROR   INFEAS.   ---- PIVOTS ----  IN      OUT
   1    1.00    0  1.000                     3.E+02 1.E+00  1  Z0      W    9
   2    1.00    9  1.000                     1.E+00 1.E+00  2  Z    9  W    1
   3    1.00   18  0.997                     9.E-01 9.E-01  1  Z    1  W   10
   4    1.00   27  0.997                     1.E+00 1.E+00  1  Z   10  W    2
   5    1.00   36  0.996                     9.E-01 4.E-01  1  Z    2  W   11
   6    1.00   45  0.996                     1.E+00 1.E+00  1  Z   11  W    6
   7    1.00   55  0.479                     2.E+00 1.E+00  1  Z    6  W    7
   8    1.00   64  0.479                     1.E+00 1.E+00  1  Z    7  W    4
   9    1.00   73  0.000                     1.E+00 1.E+00  2  Z    4  W    8
  10    1.00   73  0.000                     1.E-03 2.E-03  1  V    8  Z0

================================
Convergence Report, Iteration  1
================================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00
   1    1.14E-13    1.00E+00 (W_02    )
================================================================


Iteration   1 values.

               ROW             Z                      F
              --------    ------------          ------------
              X_01_01      2.50000E+01          -8.32667E-17
              X_01_02      3.00000E+02          -5.55112E-17
              X_01_03  L   0.00000E+00           3.60000E-02
              X_02_01      3.00000E+02          -8.32667E-17
              X_02_02  L   0.00000E+00           8.99999E-03
              X_02_03      2.75000E+02           2.77556E-17
              W_01         1.00000E+00          -1.13687E-13
              W_02         1.00000E+00           1.13687E-13
              P_01         1.22500E+00           0.00000E+00
              P_02         1.15300E+00           0.00000E+00
              P_03         1.12600E+00           0.00000E+00


Major iterations ........     1
Lemke pivots ............    10
Refactorizations ........     2
Deviation ............... 1.137E-13
Solved.


Total solution time .:        0.6 sec.
Function & Jacobian..:        0.2 sec.
LCP solution ........:        0.2 sec.
Refactorizations ....:        0.1 sec.
FTRAN ...............:        0.0 sec.
Update ..............:        0.1 sec.
```

7. **Termination Messages**

Basis factorization error in INVERT.

An unexpected error code returned by LUSOL. This should normally not occur. Examine the status file for a message from LUSOL.[11]

Failure to converge.

Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

Inconsistent parameters ZTOLZ0, ZTOLZE.

ZTOLZ0 determines the smallest value loaded into the cover vector h, whereas ZTOLZE is the feasibility tolerance employed in the Harris pivot selection procedure. If ZTOLZ0 < -ZTOLZE, Lemke's algorithm cannot be executed because the initial basis is infeasible.

Insufficient space for linearization.

Available memory is inadequate for holding the nonzeros in the Jacobian. More memory needs to be allocated. On a PC, you probably will need to install more physical memory - if there is insufficient space for the Jacobi matrix, there is far too little memory for holding the LU factors of the same matrix.

Insufficient space to invert.

More memory needs to be allocated for basis factors.

---

[11] Within GAMS, insert the line "OPTION SYSOUT=ON;" prior to the solve statement and resubmit the program in order to pass the MILES solver status file through to the listing.

Increase the value of LUSIZE in the options file, or assign a larger value to <model>.workspace if MILES is accessed through GAMS.

Iteration limit exceeded.

This can result from either exceeding the major (Newton) or minor (Lemke) iterations limit. When MILES is invoked from GAMS, the Lemke iteration limit can be set with the statement "<model>.iterlim = xx;" (the default value is 1000). The Newton iteration limit is 25 by default, and it can be modified only through the ITLIMT option.

Resource interrupt.

Elapsed CPU time exceeds options parameter RESLIM. To increase this limit, either add RESLIM = xxx in the options file or (if MILES is invoked from within GAMS), add a GAMS statement "<model>.RESLIM = xxx;".

Singular matrix encountered.

Lemke’s algorithm has been interrupted due to a singularity arising in the basis factorization, either during a column replacement or during a refactorization. For some reason, a restart is not possible.

Termination on a secondary ray.

Lemke’s algorithm terminated on a secondary ray. For some reason, a restart is not possible.

Unknown termination status.

The termination status flag has not been set, but the code

has interrupted. Look in the status file for a previous message. This termination code should not happen often.

5 **References**

K.J. Anstreicher, J. Lee and T.F. Rutherford "Crashing a Maximum Weight Complementary Basis", *Mathematical Programming*. (1992)

A. Brooke, D. Kendrick, and A. Meeraus *GAMS: A User's Guide*, Scientific Press, (1987).

R.W. Cottle and J.S. Pang *The Linear Complementarity Problem*, Academic Press, (1992).

J.E. Dennis and R.B. Schnabel *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall (1983).

S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).

B.C. Eaves, "A locally quadratically convergent algorithm for computing stationary points," Tech. Rep., Department of Operations Research, Stanford University, Stanford, CA (1978).

P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright "Maintaining LU factors of a general sparse matrix, *Linear Algebra and its Applications* 88/89, 239-270 (1991).

C.B. Garcia and W.I. Zangwill *Pathways to Solutions, Fixed Points and Equilibria*, Prentice-Hall (1981)

P. Harker and J.S. Pang "Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications", *Mathematical Programming* 48, pp. 161-220 (1990).

W.W. Hogan, "Energy policy models for project independence," *Computation and Operations Research* 2 (1975) 251-271.

N.H. Josephy, "Newton's method for generalized equations", Techncical Summary Report #1965, Mathematical Research Center, University of Wisconsin - Madison (1979).

I. Kaneko, "A linear complementarity problem with an n by 2n 'P'-matrix", *Mathematical Programming Study 7*, pp. 120-141, (1978).

C.E. Lemke "Bimatrix equilibrium points and mathematical programming", *Management Science* 11, pp. 681-689, (1965).

L. Mathiesen, "Computation of economic equilibria by a sequence of linear complementarity problems", *Mathematical Programming Study* 23 (1985).

P.V. Preckel, "NCPLU Version 2.0 User's Guide", Working Paper, Department of Agricultural Economics, Purdue University, (1987).

W.H. Press, B.P.Flannery, S.A. Teukolsky, W.T. Vetterling *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press (1986).

J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press (1970).

S.M. Robinson, "A quadratically-convergent algorithm for general nonlinear programming problems," *Mathematical Programming Study* 3 (1975).

T.F. Rutherford "Extensions of GAMS for variational and complementarity problems with applications in economic equilibrium analysis", Working Paper 92-7, Department of Economics, University of Colorado (1992a).

T.F. Rutherford "Applied general equilibrium modeling using MPS/GE as a GAMS subsystem", Working Paper 92-15, Department of Economics, University of Colorado (1992b).

Table 7 Transport Model in GAMS/MCP (page 1 of 2)

```
*==>TRNSP.GMS

option mcp=miles;

$TITLE  LP TRANSPORTATION PROBLEM FORMULATED AS A ECONOMIC EQUILIBRIUM

*       ======================================================================
*       In this file, Dantzig's original transportation model is
*       reformulated as a linear complementarity problem.  We first
*       solve the model with fixed demand and supply quantities, and
*       then we incorporate price-responsiveness on both sides of the
*       market.
*
*       T.Rutherford  3/91  (revised 5/91)
*       ======================================================================

*       This problem finds a least cost shipping schedule that meets
*       requirements at markets and supplies at factories
*
*       References:
*              Dantzig, G B., Linear Programming and Extensions
*              Princeton University Press, Princeton, New Jersey, 1963,
*              Chapter 3-3.
*
*              This formulation is described in detail in Chapter 2
*              (by Richard E. Rosenthal) of GAMS: A Users' Guide.
*              (A Brooke, D Kendrick and A Meeraus, The Scientific Press,
*              Redwood City, California, 1988.
*

  SETS
       I   canning plants   / SEATTLE, SAN-DIEGO /
       J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;

  PARAMETERS

       A(I)  capacity of plant i in cases (when prices are unity)
         /    SEATTLE     325
              SAN-DIEGO   575  /,

       B(J)  demand at market j in cases (when prices equal unity)
         /    NEW-YORK    325
              CHICAGO     300
              TOPEKA      275  /,

        ESUB(J)  Price elasticity of demand (at prices equal to unity)
         /    NEW-YORK    1.5
              CHICAGO     1.2
              TOPEKA      2.0  /;

  TABLE D(I,J)  distance in thousands of miles
                    NEW-YORK       CHICAGO      TOPEKA
      SEATTLE          2.5           1.7          1.8
      SAN-DIEGO        2.5           1.8          1.4  ;

  SCALAR F  freight in dollars per case per thousand miles  /90/ ;

  PARAMETER C(I,J)  transport cost in thousands of dollars per case ;

            C(I,J) = F * D(I,J) / 1000 ;

  PARAMETER PBAR(J) Reference price at demand node J;
```

Table 8 Transport Model in GAMS/MCP (page 2 of 2)

```
  POSITIVE  VARIABLES
       W(I)             shadow price at supply node i,
       P(J)             shadow price at demand node j,
       X(I,J)           shipment quantities in cases;

  EQUATIONS
       SUPPLY(I)        supply limit at plant i,
       FXDEMAND(J)      fixed demand at market j,
       PRDEMAND(J)      price-responsive demand at market j,
       PROFIT(I,J)      zero profit conditions;

PROFIT(I,J)..   W(I) + C(I,J)   =G= P(J);

SUPPLY(I)..     A(I) =G= SUM(J, X(I,J));

FXDEMAND(J)..   SUM(I, X(I,J))  =G=  B(J);

PRDEMAND(J)..   SUM(I, X(I,J))  =G=  B(J) * (PBAR(J)/P(J))**ESUB(J);

*       Declare models including specification of equation-variable association:

  MODEL FIXEDQTY / PROFIT.X, SUPPLY.W, FXDEMAND.P/ ;
  MODEL EQUILQTY / PROFIT.X, SUPPLY.W, PRDEMAND.P/ ;

*       Initial estimate:

  P.L(J) = 1;    W.L(I) = 1;

  PARAMETER REPORT(*,*,*)  Summary report;

  SOLVE FIXEDQTY USING MCP;
  REPORT("FIXED",I,J) = X.L(I,J);   REPORT("FIXED","Price",J) = P.L(J);
  REPORT("FIXED",I,"Price") = W.L(I);

*       Calibrate the demand functions:

  PBAR(J) = P.L(J);

*       Replicate the fixed demand equilibrium:

  SOLVE EQUILQTY USING MCP;

  REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
  REPORT("EQUIL",I,"Price") = W.L(I);

  DISPLAY "BENCHMARK CALIBRATION", REPORT;

*       Compute a counter-factual equilibrium:

  C("SEATTLE","CHICAGO") = 0.5 * C("SEATTLE","CHICAGO");

  SOLVE FIXEDQTY USING MCP;
  REPORT("FIXED",I,J) = X.L(I,J);  REPORT("FIXED","Price",J) = P.L(J);
  REPORT("FIXED",I,"Price") = W.L(I);

*       Replicate the fixed demand equilibrium:

  SOLVE EQUILQTY USING MCP;
  REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
  REPORT("EQUIL",I,"Price") = W.L(I);

  DISPLAY "Reduced Seattle-Chicago transport cost:", REPORT;
```

GAMS/MINOS

Table of Contents:

1. INTRODUCTION

This section describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver. GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems.

GAMS/MINOS is designed to find solutions that are *locally optimal*. The nonlinear functions in a problem must be *smooth* (i.e., their first derivatives must exist). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a *global optimum*. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a staring point that is "sufficiently close," but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this chapter.

2. HOW TO RUN A MODEL WITH GAMS/MINOS

MINOS is capable of solving models of the following types: LP, NLP and RMINLP. If MINOS is not specified as the default LP, NLP or RMINLP solver, then the following statement can be used in your GAMS model:

```
option lp=minos5;   { or nlp or rminlp }
```

It should appear before the `solve` statement.

3. OVERVIEW OF GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

Minimize	$F(x)$	$+ c^T x$	$+ d^T y$		*(1)*
subject to	$f(x)$		$+ A_1 y$	$<> b_1$	*(2)*
		$A_2 x$	$+ A_3 y$	$<> b_2$	*(3)*
	l	$\leq$	x,y	$\leq u$	*(4)*

where the vectors c, d, b_1, b_2, l, u and the matrices A_1, A_2, A_3 are constant, $F(x)$ is a smooth scalar function, and $f(x)$ is a vector of smooth functions. The $< >$ signs mean that individual constraints may be defined using $\leq$, $=$ or $\geq$ corresponding to the GAMS constructs =L= , =E= and =G=.

The components of x are called the nonlinear variables, and the components of y are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let m_1 and n_1 denote the number of nonlinear constraints and variables, and let m and n denote the total number of (general) constraints and variables. Thus, A_3 has m-m_1 rows and n-n_1 columns.

The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of l and u may be $-\infty$ or $+\infty$ respectively, in accordance with the GAMS use of -INF and + INF.

The vectors b_1 and b_2 are called the right-hand side, and together are denoted by b.

3.1. Linear programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program.* Since there is no need to distinguish between linear and nonlinear variables, we use x rather than y. GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\begin{array}{ll} \textit{minimize} & c^T x \\ \textit{subject to} & Ax + Is = 0 \\ & l \leq x,\ s \leq u \end{array}$$

where the elements of x are your own GAMS variables, and s is a set of *slack variables:* one for each general constraint. For computational reasons, the right-hand side b is incorporated into the bounds on s.

In the expression $Ax + Is = 0$ we write the identity matrix explicitly if we are concerned with columns of the associated matrix (AI). Otherwise we will use the equivalent notation $Ax + s = 0$.

GAMS/MINOS solves linear programs using a reliable implementation of the *primal simplex method* (Dantzig, 1963), in which the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Nx_N = 0,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B and x_N are called the basic or nonbasic variables respectively. Together they are a permutation of the vector *(x, s)*.

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equation $Bx_B = Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form.

The simplex method reaches such a solution by performing a sequence of *iterations,* in which one column of B is replaced by one column of *N*(and vice versa), until no such interchange can be found that will reduce the value of c^Tx.

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of x_B lie significantly outside their bounds, we say that the current point is *infeasible*. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function c^Tx.

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly *between* their bounds" $l_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $l_j - \delta < x_j < l_j$ or $u_j < x_j + \delta$ where δ is a *feasibility tolerance* (typically 10^{-6}). In rare cases, even a new nonbasic variables might lie outside their bounds by as much as δ.

GAMS/MINOS maintains a sparse LU factorization of the basis matrix B, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL(Gill *et al.* 1987).(see Bartels and Golub, 1969; Bartels, 1971; Reid, 1976 and 1982.) The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

3.2. Problems with a Nonlinear Objective

When nonlinearities are confined to the term F(x) in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a *reduced-gradient* algorithm (Wolfe, 1962) combined with a *quasi-Newton* algorithm follows that described in Murtagh and Saunders (1978).

In the reduced-gradient method, the constraints Ax + Is = 0 are partitioned into the form

$$Bx_B + Sx_S + Nx_N = 0$$

where x_s is a set of *superbasic variables.* At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance δ), while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be *s*, the number of columns in *S*. (The context will always distinguish s from the vector of slack variables .) At a solution, *s* will be no more than n_1, the number of nonlinear variables. In many practical cases we have found that s remains reasonably small, say 200 or less, even if n_1 is large.

In the reduced-gradient algorithm, x_s is regarded as a set of "independent variables" or "free variables" that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of *B*, *S* and *N*, some of the nonbasic variables are selected to be added to *S*, and the process is repeated with an increased value of *s*. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of *s* is reduced by one.

A step of the reduced-gradient method is called a *minor iteration.* For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix *Z* is needed now for descriptive purposes. It takes the form

$$Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}$$

though it is never computed explicitly. Given LU factorization of the basis matrix *B*, it is possible to compute products of the form Zq and Z^Tg by solving linear equations involving *B* or B^T. This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the *B*, *S*, *N* partition remains constant. A *search direction q* for the superbasic variables is obtained by solving a system of the form

$$R^TRq = -Z^Tg$$

where *g* is a gradient of *F(x)*, Z^Tg is the *reduced gradient*, and *R* is a dense upper triangular matrix. GAMS computes the gradient vector *g* analytically, using symbolic differentiation. The matrix *R* is updated in various ways in order to approximate the *reduced Hessian* according to $R^TR \approx Z^THZ$ where *H* is the matrix of second derivatives of *F(x)* (the *Hessian).*

Once *q* is available, the search direction for all variables is defined by $p = Zq$. A *line search* is then performed to find an approximate solution to the one-dimensional problem

$$\begin{array}{ll} \textit{Minimize}_{\alpha} & F(x + \alpha p) \\ \textit{subject to} & 0 < \alpha < \beta \end{array}$$

where β is determined by the bounds on the variables. Another important GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length α (see Gill *et*

at., 1979). The number of nonlinear function evaluations required may be influenced by setting the Line search tolerance, as discussed in Section D.3.

As a linear programming, an equation $B^T\pi = gB$ is solved to obtain the *dual variables* or *shadow prices* π where *gB* is the gradient of the objective function associated with basic variables. It follows that $gB - B^T\pi = 0$ The analogous quantity for superbasic variables is the reduced-gradient vector $Z^Tg = gs - s^T\pi$; this should also be zero at an optimal solution. (In practice its components will be of order $r \,||\, \pi \,||$ where *r* is the optimality tolerance, typically 10^{-6}, and $||\, \pi \,||$ is a measure of the size of the elements of π.)

3.3. Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a *project Lagrangian* algorithm, based on a method due to Robinson (1972), see Murtagh and Saunders (1982). 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.

At the start of the *k*-th major iteration, let x_k be an estimate of the nonlinear variables, and let λ_k be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing *f(x)* in equation (2) to its linear approximation:

$$f'(x, x_k) = f(x_k) + J(x_k)\,(x - x_k)$$

or more briefly

$$f' = f_k + J_k\,(x - x_k)$$

where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k. (The *i*-th row of the Jacobian is the gradient vector of the *i*-th nonlinear constraint function. As for the objective gradient, GAMS calculates the Jacobian using symbolic differentiation).

The subproblem to be solved during the *k*-th major iteration is then

$$\begin{array}{lll} \textit{Minimize}_{x,y} & F(x) + c^Tx + d^Ty - \lambda_k^T(f - f') + 0.5\,\rho\,(f - f')^T(f - f') & (5) \\ \textit{subject to} & f' \quad + A_1y \;\; <> b_1 & (6) \\ & A_2x \quad + A_3y \;\; <> b_2 & (7) \\ & l \;<\; (x,y) \;< u & (8) \end{array}$$

The objective function (5) is called an *augmented Lagrangian*. The scalar ρ is a *penalty parameter*, and the term involving ρ is a modified *quadratic penalty function*.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) - (8). As before, slack variables are introduced and b_1 and b_2 are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}+\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} s_1 \\ s_2 \end{pmatrix}=\begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}$$

This system will be referred to as $Ax + Is = 0$ as in the linear case. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_1, A_2, and A_3.

In the output from GAMS/MINOS, the term *Feasible subproblem* indicates that the *linearized constraints* have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that *feasibility* and *optimality* occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved" problems, the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iteration.

4. GAMS OPTIONS

The following GAMS options are used by GAMS/MINOS:

4.1. Options specified through the option statement

The following options are specified through the option statement. For example,

```
set iterlim = 100 ;
```

sets the iterations limit to 100.

`Iterlim`	integer	Sets the iteration limit. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. *(default = 1000)*
`Reslim`	real	Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. *(default = 1000.0)*
`Bratio`	real	Determines whether or not to use an advanced basis. *(default = 0.25)*
`Sysout`	on/off	Will echo the MINOS messages to the GAMS listing file. *(default = off)*

4.2. Options specified through model suffixes

The following options are specified through the use of the model suffix. For example,

```
mymodel.workspace = 10 ;
```

sets the amount of memory used to 10 MB. `Mymodel` is the name of the model as specified by the model statement. In order to be effective, the assignment of the model suffix should be made between the model and solve statements.

`workspace`	real	Gives MINOS x MB of workspace. Overrides the memory estimation.*(default is estimated by solver)*
`optfile`	integer	Instructs MINOS to read the option file *`minos5.opt`*. *(default = 0)*
`scaleopt`	integer	Instructs MINOS to use scaling information passed by GAMS through the *variable.SCALE* parameters*(default = 0)*

5. SUMMARY OF MINOS OPTIONS

The performance of GAMS/MINOS is controlled by a number of parameters or “options.” Each option has a default value that should be appropriate for most problems. (The defaults are given in the Section 7.) For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file.

All these options should be entered in the option file 'minos5.opt' after setting the model.OPTFILE parameter to 1. The option file is not case sensitive and the keywords must be given in full. Examples for using the option file can be found at the end of this section. The second column in the tables below contains the section where more detailed information can be obtained about the corresponding option in the first column.

5.1. Output related options

Debug level	Controls amounts of output information.
Log Frequency	Frequency of iteration log information.
Print level	Amount of output information.
Scale, print	Causes printing of the row and column-scales.
Solution No/Yes	Controls printing of final solution.
Summary frequency	Controls information in summary file.

5.2. Options affecting Tolerances

Crash tolerance	crash tolerance
Feasibility tolerance	Variable feasibility tolerance for linear constraints.
Line search tolerance	Accuracy of step length location during line search.
LU factor tolerance	Tolerance during LU factorization.
LU update tolerance	
LU Singularity tolerance	
Optimality tolerance	Optimality tolerance.
Pivot Tolerance	Prevents singularity.
Row Tolerance	Accuracy of nonlinear constraint satisfaction at optimum.
Subspace tolerance	Controls the extent to which optimization is confined to the current set of basic and superbasic variables

5.3. Options affecting Iteration limits

Iterations limit	Maximum number of minor iterations allowed
Major iterations	Maximum number of major iterations allowed.
Minor iterations	Maximum number of minor iterations allowed between successive linearizations of the nonlinear constraints.

5.4. Other algorithmic options

Check frequency	frequency of linear constraint satisfaction test.
Completion	accuracy level of sub-problem solution.
Crash option	perform crash
Damping parameter	See Major Damping Parameter
Expand frequency	Part of anti-cycling procedure
Factorization frequency	Maximum number of basis changes between factorizations.
Hessian dimension	Dimension of reduced Hessian matrix
Lagrangian	Determines linearized sub-problem objective function.
Major damping parameter	Forces stability between subproblem solutions.
Minor damping parameter	Limits the change in x during a line search.
Multiple price	Pricing strategy
Partial Price	Level of partial pricing.
Penalty Parameter	Value of ρ in the modified augmented Lagrangian.
Radius of convergence	Determines when ρ will be reduced.
Scale option	Level of scaling done on the model.
Start assigned nonlinears	Affects the starting strategy during cold start.
Superbasics limit	Limits storage allocated for superbasic variables.
Unbounded objective value	Detects unboundedness in nonlinear problems.

Unbounded step size	Detects unboundedness in nonlinear problems.
Verify option	Finite-difference check on the gradients
Weight on linear objective	Invokes the composite objective technique,

5.5. Examples of GAMS/MINOS option file

The following example illustrates the use of certain options that might be helpful for "difficult" models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

```
BEGIN GAMS/MINOS options
* These options might be relevant for very nonlinear models.
Major damping parameter  0.2   * may prevent divergence.
Minor damping parameter  0.2   * if there are singularities
                               * in the nonlinear functions.
Penalty parameter       10.0   * or 100.0 perhaps-a value
                               * higher than the default.
Scale linear variables         * (This is the default.)
END GAMS/MINOS options
```

Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the "cautious" defaults are relaxed:

```
BEGIN GAMS/MINOS options
* Suggestions for models with MILDLY nonlinear constraints
Completion   Full
Minor alteration limit   200
Penalty parameter           0.0  * or 0.1 perhaps-a value
                                 * smaller than the default.
                                 *   Scale one of the following
Scale all variables              * if starting point is VERY GOOD.
Scale linear variables           * if they need it.
Scale   No   * otherwise.
END GAMS/MINOS options
```

Most of the options described in the next section should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

6. Special Notes

6.1. Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:

- Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix .L).
- Include sensible upper and lower bounds on all variables.
- Specify a *Major damping parameter* that is lower than the default value, if the problem is suspected of being highly nonlinear.
- Specify a *Penalty parameter* ρ that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values $\lambda_k = 0$ and $\rho = 0$ for all subproblems, by specifying *Lagrangian=No*. However, convergence is much more like with the default setting, *Lagrangian=Yes*. The initial estimate of the Lagrange multipliers is then $\lambda_0 = 0$, but for later subproblems λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is $\rho = 100.0/m_1$ where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many times it is safe to specify $\rho = 0$, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

6.2. Storage

GAMS/MINOS uses one large array of main storage for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of main storage available on a particular machine, and the amount of computation time that one‘s budget and/or patience can stand.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is m, the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly 100*m* words, where one “word” is the relevant storage unit for the floating-point arithmetic being used. This usually means about 800*m* bytes for workspace. A further

300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with m general constraints requires about (*m*+300) K bytes of memory.

Another important quantity, is n, the total number of variables in *x* and *y*. The above comments assume that *n* is not much larger than *m*, the number of constraints. A typical ratio is *n/m* is 2 or 3.

If there are many nonlinear variables (i.e., if n_1 is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects s, the number of superbasic variables. Recall that *s* is zero for purely linear problems. We know that *s* is need never be larger than n_1+1. In practice, *s* is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix *R* has dimension *s* and requires about $½ s^2$ words of storage. If it seems likely that *s* will be very large, some aggregation or reformulation of the problem should be considered.

7. The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section., a sample log file is shown for for each case, and the appearing messages are explained.

7.1. Linear Programs

MINOS uses a standard two-phase Simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem [MEXLS] is as follows:

```
--- Starting compilation
--- mexls.gms(1242)
--- Starting execution
--- mexls.gms(1241)
--- Generating model ONE
--- mexls.gms(1242)
---    353 rows, 578 columns, and 1873 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .21 Mb
 Reading data...

   Itn  Nopt  Ninf  Sinf,Objective
     1     1    74  1.62609255E+01
    20    14    74  1.62609255E+01
```

```
   40     4    67  1.40246211E+01
   60    14    55  1.10065392E+01
   80    12    42  8.35651175E+00

  Itn  Nopt  Ninf  Sinf,Objective
  100     6    31  7.21650054E+00
  120    21    20  6.65998885E+00
  140    13    11  5.77113823E+00
  160     1     8  2.35537706E+00
  180     4     5  9.58810903E-01

  Itn  Nopt  Ninf  Sinf,Objective
  200     9     2  3.16184199E-01

 Itn   208 -- Feasible solution.  Objective =   4.283788540E+04

  220     2     0  4.11169434E+04
  240    25     0  3.88351313E+04
  260     1     0  3.77495457E+04
  280    11     0  3.66399964E+04

  Itn  Nopt  Ninf  Sinf,Objective
  300     2     0  3.30955215E+04
  320     2     0  3.09919632E+04
  340     5     0  2.84521363E+04
  360     1     0  2.76671551E+04
  380     2     0  2.76055456E+04
 Itn   392 --     80  nonbasics set on bound, basics recomputed

 Itn   392 -- Feasible solution.  Objective =   2.756959890E+04


 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              1         392
 Objective function       2.7569598897150E+04
 Degenerate steps              83       21.17
 Norm X,      Norm PI    2.10E+01    4.49E+04
 Norm X,      Norm PI    8.05E+03    1.06E+02  (unscaled)
--- Restarting execution
--- mexls.gms(1242)
--- Reading solution for model ONE
--- All done
```

Considering each section of the log file above in order:

```
----------------------------------------------------------------------
 Work space allocated           --     .21 Mb
 Reading data...
----------------------------------------------------------------------
```

When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem is then loaded into MINOS.

Next, the MINOS iteration log begins. In the beginning, a feasible solution has not yet been found, and the screen log during this phase of the search looks like:

```
------------------------------------ -------------------------
   Itn  Nopt  Ninf  Sinf,Objective
     1     1    74  1.62609255E+01
------------------------------------------------------------------
```

The first column gives the iteration count. A simplex iteration corresponds to a basis change. *Nopt* means "Number of non-optimalities" which refers to the number of marginals that do not have the correct sign (i.e. they do not satisfy the optimality conditions), and *Ninf* means "Number of infeasibilities" which refers to the number of infeasible constraints . The last column gives the "Sum of infeasibilities" in phase 1 (when *Ninf > 0*) or the Objective in phase 2.

As soon as MINOS finds a feasible solution, it returns a message similar to the one below that appears for the example discussed:

```
------------------------------------------------------------------
 Itn   208 -- Feasible solution.  Objective =   4.283788540E+04
------------------------------------------------------------------
```

MINOS found a feasible solution and will switch from phase 1 to phase 2. During phase 2 (or the search once feasibility has been reached), MINOS returns a message that looks like:

```
------------------------------------------------------------------
 Itn   392 --     80  nonbasics set on bound, basics recomputed
------------------------------------------------------------------
```

Even non-basic variables may not be exactly on their bounds during the optimization phase. MINOS uses a sliding tolerance scheme to fight stalling as a result of degeneracy. Before it terminates it wants to get the "real" solution. The non-basic variables are therefore reset to their bounds and the basic variables are recomputed to satisfy the constraints $Ax = b$. If the resulting solution is infeasible or non-optimal, MINOS will continue iterating. If necessary, this process is repeated one more time. Fortunately, optimality is usually confirmed very soon after the first "reset". In the example discussed, feasibility was maintained after this excercise, and the resulting log looks like:

```
------------------------------------------------------------------
 Itn   392 -- Feasible solution.  Objective =   2.756959890E+04
------------------------------------------------------------------
```

MINOS has found the optimal solution, and then exits with the following summary:

```
------------------------------------------------------------------
 Major, Minor itns              1         392
 Objective function       2.7569598897150E+04
 Degenerate steps              83       21.17
 Norm X,      Norm PI    2.10E+01    4.49E+04
 Norm X,      Norm PI    8.05E+03    1.06E+02  (unscaled)
------------------------------------------------------------------
```

For an LP, the number of major iterations is always one. The number of minor iterations is the number of Simplex iterations.

Degenerate steps are ones where a Simplex iteration did not do much other than a basis change: two variables changed from basic to non-basic and vice versa, but none of the activity levels changed. In this example there were 83 degenerate iterations, or 21.17% of the total 392 iterations.

The norms ||*x*|| , ||*pi*|| are also printed both before and after the model is unscaled. Ideally the scaled ||*x*|| should be about 1 (say in the range 0.1 to 100), and ||pi|| should be greater than 1. If the scaled ||*x*|| is closer to 1 than the final ||*x*|| , then scaling was probably helpful. Otherwise, the model might solve more efficiently without scaling.

If ||*x*|| or ||*pi*|| are several orders of magnitude different before and after scaling, you may be able to help MINOS by scaling certain sets of variables or constraints yourself (for example, by choosing different units for variables whose final values are very large.)

7.2. Nonlinear Objective

Consider the WEAPONS model from the model library which has nonlinear objective and linear constraints. The screen output resulting from running this model is as follows:

```
--- Starting compilation
--- weapons.gms(59)
--- Starting execution
--- weapons.gms(57)
--- Generating model WAR
--- weapons.gms(59)
---    13 rows, 66 columns, and 156 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .06 Mb
 Reading data...
 Reading nonlinear code...

   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
     1    -1     3  9.75000000E+01     3   64

 Itn     6 -- Feasible solution.  Objective =   1.511313490E+03

    20     5     0  1.70385909E+03    23   45
    40     3     0  1.73088173E+03    63   30
    60     1     0  1.73381961E+03   109   25
    80     1     0  1.73518225E+03   155   20

   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
   100     0     0  1.73555465E+03   201   17

 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              1         119
 Objective function       1.7355695798562E+03
 FUNOBJ, FUNCON calls         245           0
```

```
 Superbasics, Norm RG          18    2.18E-08
 Degenerate steps               2        1.68
 Norm X,      Norm PI    2.74E+02    1.00E+00
 Norm X,      Norm PI    2.74E+02    1.00E+00  (unscaled)
--- Restarting execution
--- weapons.gms(59)
--- Reading solution for model WAR
--- All done
```

The parts described in the LP example will not be repeated - only the new additions will be discussed.

```
---------------------------------------------------------------------
 Reading nonlinear code...
---------------------------------------------------------------------
```

Besides reading the matrix GAMS/MINOS also reads the instructions that allow the interpreter to calculate nonlinear function values and their derivatives.

The iteration log looks slightly different from the earlier case, and has a couple of additional columns.

```
---------------------------------------------------------------------
   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
     1    -1     3  9.75000000E+01    3   64
---------------------------------------------------------------------
```

The column *Nobj* gives the number of times MINOS calculated the objective and its gradient. The column *NSB* contains the number of superbasics. In MINOS, variables are typed basic, non-basic or superbasic. The number of superbasics once the number settles down (there may be rather more at the beginning of a run) is a measure of the nonlinearity of the problem. The final number of superbasics will never exceed the number of nonlinear variables, and in practice will often be much less (in most cases less than a few hundred).

There is not much that a modeler can do in this regard, but it is worth remembering that MINOS works with a dense triangular matrix *R* (which is used to approximate the reduced Hessian) of that dimension. Optimization time and memory requirements will be significant if there are more than two or three hundred superbasics.

Once the optimum solution has been found, MINOS exits after printing a search summary that is slightly different from the earlier case.

```
---------------------------------------------------------------------
 Major, Minor itns              1         119
---------------------------------------------------------------------
```

As in the earlier case, there is always only one major iteration. Each iteration of the reduced-gradient method is called a minor iteration.

```
---------------------------------------------------------------------
 FUNOBJ, FUNCON calls         245           0
---------------------------------------------------------------------
```

FUNOBJ is the number of times MINOS asked the interpreter to evaluate the objective function and its gradients. *FUNCON* means the same for the nonlinear constraints. As this model has only linear constraints, the number is zero.

If you know that your model has linear constraints but *FUNCON* turns out to be positive, your nonlinear objective function is being regarded as a nonlinear constraint. You can improve MINOS's efficiency by following some easy rules. Please see "Objective: special treatment of in NLP" in the index for the GAMS User's Guide.

```
------------------------------------------------------------------
 Superbasics, Norm RG          18    2.18E-08
------------------------------------------------------------------
```

The number of superbasic variables in the final solution is reported and also the norm of the reduced gradient (which should be close to zero for an optimal solution).

7.3. Nonlinear constraints

For models with nonlinear constraints the log is more complicated. CAMCGE from the model library is such an example, and the screen output resulting from running it is shown below:

```
--- Starting compilation
--- camcge.gms(444)
--- Starting execution
--- camcge.gms(435)
--- Generating model CAMCGE
--- camcge.gms(444)
---    243 rows, 280 columns, and 1356 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .36 Mb
 Reading data...
 Reading nonlinear code...

 Major Minor  Ninf Sinf,Objective    Viol    RG    NSB  Ncon Penalty    Step
 1     0     1  0.000000E+00   1.3E+02 0.0E+00  184     3 6.8E-01 1.0E+00
 2    40T   30  0.00000000E+00 8.8E+01 7.3E+02  145     4 6.8E-01 1.0E+00
 3    40T   30  0.00000000E+00 8.8E+01 1.7E+01  106     5 6.8E-01 1.0E+00
 4    40T   30  0.00000000E+00 8.8E+01 2.1E+01   66     6 6.8E-01 1.0E+00
 5    40T   25  0.00000000E+00 8.8E+01 1.7E+01   26     7 6.8E-01 1.0E+00
 6    29     0  1.91734197E+02 2.6E-03 0.0E+00    0     9 6.8E-01 1.0E+00
 7     0     0  1.91734624E+02 2.3E-08 0.0E+00    0    10 1.4E+00 1.0E+00
 8     0     0  1.91734624E+02 4.5E-13 0.0E+00    0    11 1.4E+00 1.0E+00
 9     0     0  1.91734624E+02 6.8E-13 0.0E+00    0    12 0.0E+00 1.0E+00

 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              9         189
 Objective function       1.9173462423688E+02
 FUNOBJ, FUNCON calls           8          12
 Superbasics, Norm RG           0    0.00E+00
 Degenerate steps               0         .00
```

```
 Norm X,      Norm PI    1.37E+03    2.88E+01
 Norm X,      Norm PI    8.89E+02    7.01E+01  (unscaled)
 Constraint violation    6.82E-13    7.66E-16
--- Restarting execution
--- camcge.gms(444)
--- Reading solution for model CAMCGE
--- All done
```

Note that the iteration log is different from the two cases discussed above. A small section of this log is shown below.

```
----------------------------------------------------------------------
Major Minor  Ninf  Sinf,Objective    Viol    RG    NSB  Ncon Penalty Step
1     0      1  0.00000000E+00 1.3E+02 0.0E+00  184     3 6.8E-01 1.0E+00
2    40T   30  0.00000000E+00 8.8E+01 7.3E+02  145     4 6.8E-01 1.0E+00
----------------------------------------------------------------------
```

Two sets of iterations - Major and Minor, are now reported. A description of the various columns present in this log file follows:

Major	A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.
Minor	The number of minor iterations performed on the linearized subproblem. If it is a simple number like 29, then the subproblem was solved to optimality. Here, *40T* means that the subproblem was terminated. (There is a limit on the number of minor iterations per major iteration, and by default this is 40.) in general the *T* is not something to worry about. Other possible flags are *I* and *U*, which mean that the subproblem was Infeasible or Unbounded. MINOS may have difficulty if these keep occurring.
Ninf	The number of infeasibilities at the end of solving the subproblem. It is 0 if the linearized constraints were satisfied. It then means that all linear constraints in the original problem were satisfied, but only *Viol* (below) tells us about the nonlinear constraints.
Objective	The objective function for the original nonlinear program.
Viol	The maximum violation of the nonlinear constraints.
RG	The reduced gradient for the linearized subproblem. If *Viol* and *RG* are small and the subproblem was not terminated, the original problem has almost been solved.
NSB	The number of superbasics. If *ninf > 0* at the beginning, it will not matter too much if *NSB* is fairly large. It is only when *ninf* reaches 0 that MINOS starts working with a dense matrix *R* of dimension *NSB*.

Ncon — The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

Penalty — The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

Step — The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. The quantity *Viol* should then decrease rapidly. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the *Major Damping parameter*.

The final exit summary has an additional line in this case. For the problem being described, this line looks like:

```
------------------------------------------------------------------
 Constraint violation    6.82E-13    7.66E-16
------------------------------------------------------------------
```

The first number is the largest violation of any of the nonlinear constraints (the final value of *Viol*). The second number is the relative violation, $Viol/(1.0 + ||x||)$. It may be more meaningful if the solution vector x is very large.

For an optimal solution, these numbers must be small.

Note: The CAMCGE model (like many CGE models or other almost square systems) can better be solved with the MINOS option *Start Assigned Nonlinears Basic*. It can be noticed in the log that the algorithm starts out with lots of super basics (corresponding to the non-linear variables). MINOS starts with removing these superbasics, and only then starts working towards an optimal solution.

---

8. Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. The letters I and r denote integer and real values. The number ε denotes machine precision (typically 10^{-15} or 10^{-16}). Options not specified will take the default values shown.

Check frequency i — Every *i*-th iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = 0$ where s is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is

computed. If the largest component of *r* is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

(*Default = 30*)

Completion Full/Partial

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion), or to full accuracy (full completion), GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems.

Use of partial completion may reduce the work during early major iterations, unless the *Minor iterations* limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion.

An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below *100*(Row tolerance)*, the relative change in λ_k is 0.1 or less, and the previous subproblem was solved to optimality.

Full completion tends to give better Langrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

(*Default = FULL*)

Crash option i

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix (*AI*). The value of *i* determines which columns of A are eligible. Columns of *I* are used to fill “gaps” where necessary.

If *i > 0*, three passes are made through the relevant columns of *A*, searching for a basis matrix that is essentially triangular. A column is assigned to “pivot” on a particular row if the column contains a suitably large element in a row that has not yet been assigned.(The pivot elements ultimately form the diagonals of the triangular basis).

Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality (=E=) constraints. Pass 3 allows the pivots to be in inequality rows.

For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

(*Default = 1*)

0	The initial basis will contain only slack variables: $B = I$
1	All columns of A are considered (except those excluded by the *Start assigned nonlinears* option).
2	Only the columns of A corresponding to the linear variables y will be considered.
3	Variables that appear nonlinearly in the objective will be excluded from the initial basis.
4	Variables that appear nonlinearly in the constraints will be excluded from the initial basis.

Crash tolerance r

The *Crash tolerance r* allows the starting procedure *CRASH* to ignore certain "small" nonzeros in each column of *A*. If a_{max} is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|aij| < a_{max}.r$. To be meaningful, *r* should be in the range $0 \leq r < 1$).

When r > 0.0 the basis obtained by *CRASH* may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of *A* and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A are the matrix shown under *LU factor tolerance* ; i.e., a tridiagonal matrix entries -1, 4, -1. To help *CRASH* choose all m columns for the initial basis, we could specify *Crash tolerance r* for some value of $r > 0.25$.

(*Default = 0.1*)

Damping parameter r

See Major Damping Parameter

(*Default = 2.0*)

Debug level i

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness.

(*Default = 0*)

0	No debug output.
2 (or more)	Output from *M5SETX* showing the maximum residual after a row check.
40	Output from *LU8RPC* (which updates the LU factors of the

basis matrix), showing the position of the last nonzero in the transformed incoming column

50 Output from *LU1MAR* (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.

100 Out from *M2BFAC* and *M5LOG* listing the basic and superbasic variables and their values at every iteration.

Expand frequency i This option is part of anti-cycling procedure designed to guarantee progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is δ.

Over a period of i iterations, the tolerance actually used by GAMS/MINOS increases from 0.5 δ to δ (in steps $0.58\ \delta / i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing *i* helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see *Pivot tolerance*).

(*Default = 10000*)

Factorization frequency i At most i basis changes will occur between factori-zations of the basis matrix.

With linear programs, the basis factors are usually updated every iteration. The default i is reasonable for typical problems. Higher values up to i =100 (say) may be more efficient on problems that are extremely sparse and well scaled.

When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made

regularly (according to the *Check frequency*) to ensure that the general constraints are satisfied. If necessary the basis will be re-factorized before the limit of i updates is reached.

When the constraints are nonlinear, the Minor iterations limit will probably preempt i.

(*Default = 50*)

Feasibility tolerance r

When the constraints are linear, *a feasible solution* is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance *r*. (Since slacks are included, this means that the general linear constraints are also satisfied to within r.)

GAMS/MINOS attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let *SINF* be the corresponding sum of infeasibilities, If *SINF* is quite small, it may be appropriate to raise *r* by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

If *SINF* is not small, there may be other points that have a significantly smaller sum of infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum.

If *Scale option = 1* or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful)

A nonlinear objective function *F(x)* will be evaluated only at feasible points. If there are regions where *F(x)* is undefined, every attempt should be made to eliminate these regions from the problem. For example, for a function *F(x) = sqrt(x_1) + log(x_2)*, it should be essential to place lower bounds on both variables. If *Feasibility tolerance = 10^{-6}*, the bounds $x_1 > 10^{-5}$ and $x_2 > 10^{-4}$ might be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.)

If the constraints are nonlinear, the above comments apply to each major iteration. A “feasible solution” satisfies the current linearization of the constraints to within the tolerance *r*. The associated subproblem is said to be feasible.

As for the objective function, bounds should be used to keep *x* more than *r* away from singularities in the constraint functions *f(x)*.

At the start of major iteration *k*, the constraint functions $f(x_k)$

are evaluated at a certain point x_k. This point always satisfies the relevant bounds ($l < x_k < u$), but may not satisfy the general linear constraints.

During the associated minor iterations, *F(x)* and *f(x)* will be evaluated only at points x that satisfy the bound and the general linear constraints (as well as the linearized nonlinear constraints).

If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily, in several stages.

Feasibility with respect to the nonlinear constraints themselves is measured against the *Row tolerance* (not against *r*. The relevant test is made at the *start* of a major iteration.

(*Default = 1.0E-6*)

Hessian dimension r

This specifies than an *r X r* triangular matrix *R* is to be available for use by the quasi-Newton algorithm (to approximate the reduced Hessian matrix according to $Z^THZ \approx R^TR$. Suppose there are s superbasic variables at a particular iteration. *Whenever possible, r should be greater than s.*

If $r > s$, the first *s* columns of *R* will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear.

If $r < s$, a matrix of the form,

$$R = \begin{pmatrix} R_r & 0 \\ & D \end{pmatrix}$$

will be used to approximate the reduced Hessian, Where R_r is an *r X r* upper triangular matrix and *D* is a *diagonal* matrix of order *s - r*. The rate of convergence will no longer be superlinear (and may be arbitrarily slow.

The storage required if of the order $½ r^2$, which is substantial if *r* is as large as 200 (say). In general, *r* should be slight over-estimate of the final number of superbasic variables, whenever storage permits. It need not be larger than n_1+1, where n_1 is the number of nonlinear variables. For many problems it can be much smaller than n_1.

If *Superbasics limit s* is specified, the default value of *r* is the same number, *s* (and conversely). This is a safeguard to

ensure super-linear convergence wherever possible. If neither *r* nor *s* is specified, GAMS chooses values for both, using certain characteristics of the problem.

(*Default = Superbasics limit*)

Iterations limit i

This is maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the *GAMS ITERLIM* specification. If i = 0, no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

Iters or *Itns* are alternative keywords.

(*Default = 1000*)

Lagrangian Yes/No

This determines the form of the objective function used for the linearized subproblems. The default value *yes* is highly recommended. The *Penalty parameter* value is then also relevant.

If *No* is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not occur.

(*Default = yes*)

Linesearch tolerance r

For nonlinear problems, this controls the accuracy with which a step-length α is located in the one-dimensional problem

$$\begin{array}{ll} \textit{Minimize}_{\alpha} & F(x + \alpha p) \\ \textit{subject to} & 0 < \alpha \leq \beta \end{array}$$

A linesearch occurs on most minor iterations for which *x* is feasible. [If the constraints are nonlinear, the function being minimized is the augmented Lagrangian in equation (5).]

r must be a real value in the range $0.0 < r < 1.0$.

The default value $r = 0.1$ requests a moderately accurate search. It should be satisfactory in most cases.

If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try $r = 0.01$ or $r = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints.

If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try *r = 0.5* or perhaps *r = 0.9*. (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

(*Default = 0.1*)

Log Frequency i

In general, one line of the iteration log is printed every *i*-th minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions,

A value such as *i = 10, 100* or larger is suggested for those interested only in the final solution

Log frequency 0 may be used as shorthand for *Log frequency 99999*.

If *Print level > 0*, the default value of *i* is 1. If *Print level = 0*, the default value of *i* is 100. If *Print level = 0* and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

(*Default = 1 or 100*)

LU factor tolerance r_1
LU update tolerance r_2
LU Singularity tolerance r_3

The first two tolerances affect the stability and sparsity of the basis factorization $B = LU$ during re-factorization and updates respectively. The values specified must satisfy $r_i \geq 1.0$. The matrix L is a product of matrices of the form.

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix}$$

where the multipliers μ will satisfy $|\mu| < r_i$.

1. The default values $r_i = 10.0$ usually strike a good compromise between stability and sparsity.
2. For large and relatively dense problems, $r_i = 25.0$ (say) may give a useful improvement it sparsity without impairing stability to a serious degree.
3. For certain very regular structures (e.g., band matrices) it may be necessary to set and r_1 and/or r_2 to values smaller than the default in order to achieve stability. For example, if

the columns of *A* include a sub-matrix of the form

$$\begin{pmatrix} 4 & -1 & \\ -1 & 4 & -1 \\ & -1 & 4 \end{pmatrix}$$

it would be judicious to set both r_1 and r_2 to values in the range $1.0 < r_i < 4.0$. The singularity tolerance r_3 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \le r_3$ or $|U_{jj}| < r_3 \, max_i \, |U_{jj}|$, the j-th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases , the Jacobian matrix may converge to values that make the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting $r_3 = 1.0E\text{-}5$, say, may help cause a judicious change of basis.

(*Default values:* $r_1 = 10.0$, $r_2 = 10.0$, $r3 = \varepsilon^{2/3} \approx 10^{-11}$)

Major damping parameter r

The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions (x_k, λ_k) and (x_{k+1}, λ_{k+1}). For example, the default value 2.0 prevents the relative change in either x_k or λ_k from exceeding 200 percent. It will not be active on well behaved problems.

The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus x_{k+1} and λ_{k+1} are changed to

$$x_k + \sigma(x_{k+1} - x_k) \text{ and } \lambda_k + \sigma(\lambda_{k+1} - \lambda_k)$$

for some step-length $\sigma < 1$. In the case of nonlinear equation (where the number of constraints is the same as the number of variables) this gives a *damped Newton method*.

This is very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher *Penalty parameter* (say 10 or 100 times the default ρ). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of ρ is irrelevant.)

If the subproblem solutions continue to change violently, try reducing *r* to 0.2 or 0.1 (say).

For implementation reason, the shortened step to σ applies to the nonlinear variables x, but not to the linear variables *y* or the slack variables *s*. This may reduce the efficiency of the control.

(*Default = 2.0*)

Major iterations i

This is maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations my not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

(*Default = 50*)

Minor damping parameter r

This parameter limits the change in *x* during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

A linesearch of the form $minimize_{\alpha}\ F(x + \alpha p)$ is performed over the range $0 < \alpha \leq \beta$, where β is the step to the nearest upper or lower bound on *x*. Normally, the first step length tried is $\alpha_1 = min\ (1,\beta)$.

In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of *r* can lean to floating-point overflow. The parameter *r* is therefore used to define a limit

$$\beta' = r\ (1 + ||x||)/||p||$$

and the first evaluation of *F(x)* is at the potentially smaller steplength $\alpha_1 = min(1, \beta, \beta')$.

Wherever possible, upper and lower bounds on *x* should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value *r = 2.0* shout not affect progress on well behaved problems, but setting *r = 0.1 or 0.01* may be helpful when rapidly varying function are present. A "good" starting point may be required. An important application is to the class of nonlinear least squares problems.

In case where several local optima exist, specifying a small value for *r* may help locate an optima near the starting point.

(*Default = 2.0*)

Minor iterations i

This is the maximum number of minor iterations allowed between successive linearizations of the nonlinear constraints.

A moderate value (e.g., $20 \leq i \leq 50$) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion.

The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations, (say *K*) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify > *K*, to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. (If a "good" initial subproblem is arbitrarily interrupted by a small I th subsequent linearization may be less favorable than the first.)

In general it is unsafe to specify value as small as *i* = 1or 2.)even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal.)

The *Iteration limit* provides an independent limit on the total minor iterations (across all subproblems).

If the constraints are linear, only the *Iteration limit* applies: the *minor iterations* value is ignored.

(*Default = 40*)

Multiple price i

"pricing" refers to a scan of the current non-basic variables to determine if any should be changed from their value (by allowing them to become superbasic or basic).

If multiple pricing in effect, the *i* best non-basic variables are selected for admission of appropriate sign.) If partial pricing is also in effect , the best *i* best variables are selected from the current partition of *A* and *I*.

The default *i = 1* is best for linear programs, since an optimal solution will have zero superbasic variables.

Warning : If *i > 1*, GAMS/MINOS will use the *reduced-gradient method* (rather than the simplex method) even on purely linear problems. The subsequent iterations of *not* correspond to the efficient "minor iterations" carried out be commercial linear programming system using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving *i* dense columns of dimension *m*, and *i* is therefore limited for storage reasons

typically to the range $2 \leq i \leq 7$.)

GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of i . Larger values of i are therefore practical, but in general the iterations and time required when $i > 1$ are greater than when the simplex method is used ($i=1$) .

On large nonlinear problems it may be important to set $i > 1$ if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use $i = 10$ (say) for early runs, until it seems that the number of superbasics has leveled off.

If *Multiple price i* is specified , it is also necessary to specify *Superbasic limit s* for some $s > i$.

(*Default = 1*)

Optimality tolerance r

This is used to judge the size of the reduced gradients $d_j = g_j - \pi^T a_j$, where g_j is the gradient of the objective function corresponding to the j-th variable. a_j is the associated column of the constraint matrix (or Jacobian), and π is the set of dual variables.

By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy

$$dj / ||\pi|| \geq -r \quad or \quad dj / ||\pi|| \leq r$$

respectively, and if $dj / ||\pi|| \leq r$ for superbasic variables.

In the $||\pi||$ is a measure of the size of the dual variables. It is included to make the tests independents of a scale factor on the objective function.

The quantity actually used is defined by

$$\sum_{i=1} \quad , \qquad \| \pi \| = max\{ \sigma / \sqrt{m}, 1 \}$$

so that only large scale factors are allowed for.

If the objective is scaled down to be *small*, the optimality test effectively reduced to comparing D_j against r.

(*Default = 1.0E-6*)

Partial Price i

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (when a nonbasic variable is selected to become basic or superbasic).

When $i = 1$, all columns of the constraints matrix (A I) are searched.

Otherwise, A_j and I are partitioned to give i roughly equal segments A_j , I_j $(j = 1$ *to* $i)$. If the previous search was successful on A_{j-1} , I_{j-1}, the next search begins on the segments A_j , I_j. (All subscripts here are modulo i.)

If a reduced gradient is found that is large than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (several may be selected if multiple pricing has been specified.) IF nothing is found, the search continues on the next segments $A_j + 1$, $I_j + 1$ and so on.

Partial price t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

(*Default = 10 for LPs, or 1 for NLPs*)

Penalty Parameter r

This specifies the value of ρ in the modified augmented Lagrangian. It is used only when *Lagrangian = yes* (the default setting).

For early runs on a problem is known to be unknown characteristics, the default value should be acceptable. If the problem is problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of ρ may be necessary of known to ensure convergence, *even convex programs.*

On the other hand, if ρ is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify *penalty parameter 0.0*

Initially, use a moderate value for r (such as the default) and a reasonably low *Iterations* and/or *major iterations* limit.

If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the *Major damping parameter*.)

If there appears to be little progress between major iteration, it may help to reduce the penalty parameter.

(*Default = 100.0/*m_1)

Pivot Tolerance r

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular. The default value of *r* should be satisfactory in most circumstances.

When x changes to $x + \alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.

For linear problems, elements of P are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.

For nonlinear problems, elements smaller than $r\ ||p||$ are ignored.

It is common (on "degenerate" problems) for two or more variables to reach a bound at essentially the same time. In such cases, the *Feasibility tolerance* (say *t*) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of *t* should not be specified.

To a lesser extent, the *Expand frequency* (say *f*) also provides some freedom to maximize pivot the element. Excessively large values of *f* should therefore not be specified .

(*Default* = $\varepsilon^{2/3} \approx 10^{-11}$)

Print level i

This varies the amount of information that will be output during optimization.

Print level 0 sets the default *Log* and *summary frequencies* to 100. It is then easy to monitor the progress of run.

Print level 1 (or more) sets the default *Log* and *summary frequencies* to 1, giving a line of output for every minor iteration. *Print level 1* also produces basis statistics., i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized.

For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The value of i is best thought of as a binary number of the form

Print level *JFLXB*

where each letter stand for a digit that is either 0 or 1. The quantities referred to are:

B — Basis statistics, as mentioned above.

X — x_k, the nonlinear variables involved in the objective function or the constraints.

L — λ_k, the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if *Lagrangian=No*, since then $\lambda_k=0$.)

F — $f(x_k)$, the values of the nonlinear constraint functions.

J — $J(x_k)$, the Jacobian matrix.

To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, *Print level 10* sets *X=1* and the other digits equal to zero; the nonlinear *variables* will be printed each major iteration.

If *J =1*, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if *J = 1*, there is no reason to specify *X = 1* unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3   1.250000D+01   BS   1   1.00000D+00
4   2.00000D+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually different from the GAMS row numbers; see the *Solution* option.)

(*Default = 0*)

Radius of convergence r

This determines when the penalty parameter ρ will be reduced (if initialized to a positive value). Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multiplier estimate must be less than *r* at the start of a major iteration before ρ is reduced or set to zero.

A few major iterations later, full completion will be requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

(*Default = 0.01*)

Row Tolerance r

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about that an accuracy.

Let *ROWERR* be the maximum component of the residual vector $f(x) + A_1y - b_1$, normalized by the size of the solution. Thus

$$ROWERR = || f(x) + A_1y - b_1||_\infty / (1+XNORM)$$

Where *XNORM* is a measure of the size of the current solution (x,y). The solution is regarded acceptably feasible if $ROWERR \leq r$.

If the problem functions involve data that is known to be of low accuracy, a larger *Row tolerance* may appropriate.

(*Default = 1.0E-6*)

Scale option i

Scaling done on the model.

(*Default = 2 for LPs, 1 for NLPs*)

0

No scaling. If storage is at a premium, this option should be used

1

Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer, 1982). This will sometimes improve the performance of the solution procedures. *Scale linear variables* is an equivalent option.

2

All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side *b* or the solution *x* is large. This takes into account columns of (*AI*) that are fixed or have positive lower bounds or negative upper bounds. *Scale nonlinear variables* or *Scale all variables* are equivalent options.

Scale Yes sets the default. (*Caution*: If all variables are nonlinear, *Scale Yes* unexpectedly does nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent to *Scale Option 0*).

If nonlinear constraints are present, *Scale option 1* or *0* should generally be rid at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should therefore be used only if (a) good starting point is provided, and (b) the problem is not highly nonlinear.

Scale, print

This causes the row-scales *r(i)* and column-scales *c(j)* to be printed. The scaled matrix coefficients are $a'_{ij} = a_{ij}\, c(j)/r(i)$, and the scaled bounds on the variables, and slacks are $l'_j = l_j/c(j)$, $u'_j = u_j/c(j)$, where $c(j) = r(j-n)$ if $j > n$.

If a *Scale option* has not already been specified, *Scale, print* sets the default scaling.

Scale Tolerance r

All forms except *Scale option* may specify a tolerance *r* where $0 < r < 1$ (for example: *Scale, Print, Tolerance = 0.99*). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ration of the largest and smallest nonzero coefficients in each column:

$$\rho_j = max_i\, |a_{ij}| \,/\, min_i\, |a_{ij}| \;\; (a_{ij} \neq 0)$$

If $max_j\ \rho_j$ is less than *r* times its previous value, another scaling pass is performed to adjust the row and column scales. Raising *r* from 0.9 to 0.99 (say) usually increases the number of scaling passes through *A*. At most 10 passes are made.

If a *Scale option* has not already been specified, *Scale tolerance* sets the default scaling.

(*Default = 0.9*)

Solution No/Yes

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS "column numbers," and those at the right of each line in the rows section are GAMS/MINOS "slacks".)

The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

(*Default = No*)

Start assigned nonlinears

This option affects the starting strategy when there is no basis (i.e., for the first solve or when *option bratio = 1* is used to reject an existing basis.)

This option applies to all nonlinear variables that have been assigned non-default initial values and are strictly between their bounds. Free variables at their default value of zero are

excluded. Let *K* denote the number of such "assigned nonlinear variables."

Note that the *first* and *fourth* keywords are significant.

(*Default = superbasic*)

Superbasic — Specify *superbasic* for highly nonlinear models, as long as *K* is not too large (say *K < 100)* and the initial values are "good".

Basic — Specify *basic* for models that are essentially "square" (i.e., if there are about as many general constraints as variables).

Nonbasic — Specify *nonbasic* if *K* is large.

Eligible for crash — Specify *eligible for Crash* for linear or nearly linear models. The nonlinear variables will be treated in the manner described under *Crash* option.

Subspace tolerance r — This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3).

r must be a real number in the range $0 < r \leq 1$.

When a nonbasic variables x_i is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $||Z^T g_0||$. (In fact, the norm will be $|d_i|$, the size of the reduced gradient for the new superbasic variable x_j .

Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies $\|Z^T g_0\| \leq r$ $\|Z^T g_0\|$ is the size of the largest reduced-gradient component among the superbasic variables.)

A smaller value of *r* is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as *r* = 0.9 may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution.

Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of *r*.

(*Default = 0.5*)

Summary frequency i — A brief form of the iteration log is output to the summary

file. In general, one line is output every *i*-th minor iteration., In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated.

The *Summary frequency* controls summary output in the same as the *log frequency* controls output to the print file

A value such as *i* = 10 or 100 is often adequate to determine if the SOLVE is making progress. If *Print level = 0*, the default value of *i* is 100. If *Print level > 0*, the default value of *i* is 1. If *Print level = 0* and the constraints are nonlinear, the *Summary frequency* is ignored. Instead, one line is printed at the beginning of each major iteration.

(*Default = 1 or 100*)

Superbasics limit i

This places a limit on the storage allocated for superbasic variables. Ideally, *i* should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.

For linear problems, an optimum is normally a basic solution with no degrees of freedom.(The number of variables lying strictly between their bounds is not more than *m*, the number of general constraints.) The default value of *i* is therefore 1.

For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables."

Normally, *i* need not be greater than n_1+1, where n_1 is the number of nonlinear variables.

For many problems, *i* may be considerably smaller than n_1. This will save storage if n_1 is very large.

This parameter also sets the *Hessian dimension*, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

(*Default = Hessian dimension*)

Unbounded objective value r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If $|F|$ exceeds r or if α exceeds r_2, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`.

If singularities are present, unboundedness in *F(x)* may be

manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against r_1 can be made.

Unboundedness is x is best avoided by placing finite upper and lower bounds on the variables. see also the *Minor damping parameter*.

(*Default = 1.0E+20*)

Unbounded step size r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If α exceeds r, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`.

If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against r_1 can be made.

Unboundedness is x is best avoided by placing finite upper and lower bounds on the variables. see also the *Minor damping parameter*.

(*Default = 1.0E+10*)

Verify option i

This option refers to a finite-difference check on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as "correct".

Gradient verification occurs before the problem is scaled, and before the first basis is factorized. (Hence, it occurs before the basic variables are set to satisfy the general constraints $Ax + s = 0$.)

(*Default = 0*)

0 Only a "cheap" test is performed, requiring three evaluations of the nonlinear objective (if any) and two evaluations of the nonlinear constraints. *Verify No* is an equivalent option.

1 A more reliable check is made on each component of the objective gradient. *Verify objective gradients* is an equivalent option.

2 A check is made on each column of the Jacobian matrix

	associated with the nonlinear constraints. *Verify constraint gradients* is an equivalent option.
3	A detailed check is made on both the objective and the Jacobian. *Verify*, *Verify gradients*, and *Verify Yes* are equivalent options.
-1	No checking is performed.
Weight on linear objective r	The keywords invokes the so-called *composite objective* technique, if the first solution obtained infeasible, and if the objective function contains linear terms. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear objective. At each infeasible iteration, the objective function is defined to be $minimize_x \ \sigma w(c^T x) + (sum\ of\ infeasibilities)$ where $\sigma = 1$ for minimization and $\sigma = -1$ for maximization and c is the linear objective. If an "optimal" solution is reached while still infeasible, *w* is reduced by a factor of 10. This helps to allow for the possibility that the initial *w* is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending towards zero. The effect of *w* is disabled after five such reductions, or if a feasible solution is obtained. This option is intended mainly for linear programs. It is unlikely to helpful if the objective function is nonlinear. (*Default = 0.0*)

9. Acknowledgements

This chapter is a minor revision to the earlier description of the GAMS/MINOS system found in the GAMS User's Guide. Phillip Gill, Walter Murray, Bruce A. Murtagh, Michael A. Saunders, and Margaret H. Wright were the authors of this earlier description. The section on the GAMS/MINOS log file was initially put together by Michael Saunders and Arne Drud.

10. References

Bartels, R.H. (1971), A stabilization of the simplex method, Numerische Mathematik 16, 414-434.

Bartels, R.H. and G. H. Golub (1969), The simplex method of linear programming using the LU decomposition, *Communications of the ACM* 12, 266-268.

Dantzig, G.B. (1963), *Linear Programming and Extensions,* Princeton University Press, Princeton, New Jersey.

Davidon, W.C. (1959), Variable metric methods for minimization, A.E.C. Res. and Develop. Report ANL-5990, Argonne National Laboratory, Argonne, IL.

Fourer, R. (1982), Solving staircase linear programs by the simplex method, *Mathematical Programming* 23, 274-313.

Gill, P.E. , W, Murray, M. A. Saunders and M. H. Wright (1979), Two step-length algorithms for numerical optimization, Report SOL 79-25, Department of Operations Research, Stanford University, Stanford, California

Gill, P.E. , W, Murray, M. A. Saunders and M. H. Wright (1987), Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications* 88/89, 239-270.

Murtagh, B.A. and M. A. Saunders (1978), Large-scale linearly constrained optimization, *Mathematical Programming* 14, 41-72.

Murtagh, B.A. and M. A. Saunders (1982), A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Mathematic Programming Study* 16, *Algorithm for Constrained Minimization of Smooth Nonlinear Function,* 84-117.

Murtagh, B.A. and M. A. Saunders (1983), MINOS 5.0 User‘s Guide, Report SOL 83-20, Department of Operations Research, Stanford University, (Revised as MINOS 5.1 User‘s Guide, Report SOL 83-20R, 1987.)

Reid, J.K. (1976), Fortran subroutines for handling sparse linear programming bases, Report R8269, Atomic Energy Research Establishment, Harwell, England.

Reid, J.K. (1982), A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases, *Mathematical Programming* 24, 55-69.

Robinson, S,M. (1972), A quadratically convergent algorithm for general nonlinear programming problems, *Mathematical Programming* 3, 145-156.

Wolfe, P (1962), The reduced-gradient method, unpublished manuscript, RAND Corporation.

GAMS / MPSWRITE 1.2

Release Notes

MPSWRITE 1.2 – February 22, 2000 – Differences from 1.1

- Enabled the use of long identifier and label (set element) names. In earlier versions, the length of identifier and element names was limited to 10 characters.
- Added the option OBJLAST (OBJective row LAST) to place the objective row as the last/first row in the MPS matrix. Most solution systems have MPS file readers which take the first free row (N) to be the objective row. Previous versions of MPSWRITE placed the objective row as the last row in the matrix.

MPSWRITE 1.1 - April 15, 1999 – Differences from 1.0

- Updated the ANALYZE link to be compatible with the book by Harvey J. Greenberg, *A Computer-Assisted Analysis for Mathematical Programming Models and Solutions: A User's Guide to ANALYZE*, Kluwer Academic Publishers, Boston, 1993.
- Added the option MCASE (Mixed CASE names) to control the casing of MPS names.

1 Introduction

The GAMS modeling language allows you to represent a mathematical programming model in a compact and easily understood form and protects the user from having to represent the model in the form that is required by any particular solver. This solver specific form is usually difficult to write and even more difficult to read and understand. However, there may be cases in which you require the model data in precisely such a form. You may need to use applications or solvers that are not interfaced to GAMS or you like to test developmental algorithms either in an academic or industrial setting. MPSWRITE is a subsystem of GAMS that addresses this problem. It is not a solver and can be used independently of them. It can generate the following files for a GAMS model:

- MPS matrix file
- MPS basis file
- GAMS to MPS mapping file
- ANALYZE syntax file
- ANALYZE LP file

Many solvers can read an optimization problem in the form of an MPS file. The MPS matrix file stores the problem matrix as well as the various bounds in a specified format. In this file, names with up to eight characters are used to specify the variables and constraints. In GAMS, however, the variable or constraint name can have as many hundreds of characters. Therefore, in order to create the MPS file, MPSWRITE must generate unique names with a maximum of eight characters for the variable and constraint names.

The MPS basis file describes the status of all the variables and constraints at a point. It is also written in the MPS format. The mapping file relates the names generated for the MPS matrix and basis files to their corresponding GAMS name.

Since the MPS file format can only handle linear models, nonlinear models are linearized around the current point. The matrix coefficients are then the partial derivatives of the nonlinear terms and the right-hand-sides are adjusted accordingly.

MPSWRITE is capable of generating MPS files for models of the following types; `lp`, `rmip`, `mip`, `nlp`, `dnlp`, `rminlp`, `minlp`. In case of NLP models, the MPS file is generated at the current point.

ANALYZE is a system developed by Harvey Greenberg and coworkers at the University of Colorado - Denver for the analysis of linear programming models and solutions. The LP file contains the matrix and solution information for the model while the syntax file is similar in role to the mapping file but in a form recognizable by ANALYZE.

2 AN EXAMPLE

In order to illustrate the use of MPSWRITE add the following lines to [TRNSPORT] just before the `solve` statement

```
option lp = mpswrite ;
```

or simply change the default LP solver by using a GAMS command line parameter

```
>gams trnsport lp=mpswrite
```

Although we use the `solve` statement to execute MPSWRITE, no optimization will take place. The current solution values will remain unchanged. By default the MPSWRITE outputs the MPS file, the basis file and the mapping file. The generation of any of these files can be turned off using options in the option file as will be explained later.

Apart from the listing file that re1ports model statistics, the following files are also generated: by running the model - *`gams.mps`*, `gams.bas`, `gams.map`.

```
NAME          GAMSMOD
* OBJECTIVE ROW HAS TO BE MINIMIZED
* ROWS       :        7
* COLUMNS    :        7
* NON-ZEROES :       20
ROWS
 N  R0000000
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
COLUMNS
    C0000001  R0000001  -0.225000000
    C0000001  R0000002   1.000000000
    C0000001  R0000004   1.000000000
    C0000002  R0000001  -0.153000000
    C0000002  R0000002   1.000000000
    C0000002  R0000005   1.000000000
    C0000003  R0000001  -0.162000000
    C0000003  R0000002   1.000000000
    C0000003  R0000006   1.000000000
    C0000004  R0000001  -0.225000000
    C0000004  R0000003   1.000000000
    C0000004  R0000004   1.000000000
    C0000005  R0000001  -0.162000000
    C0000005  R0000003   1.000000000
    C0000005  R0000005   1.000000000
    C0000006  R0000001  -0.126000000
    C0000006  R0000003   1.000000000
    C0000006  R0000006   1.000000000
    C0000007  R0000001   1.000000000
    C0000007  R0000000   1.000000000
```

```
RHS
    RHS       R0000002   350.0000000
    RHS       R0000003   600.0000000
    RHS       R0000004   325.0000000
    RHS       R0000005   300.0000000
    RHS       R0000006   275.0000000
BOUNDS
 FR BOUND     C0000007
ENDATA
```

Figure 1: MPS matrix file for trnsport.gms

```
NAME          GAMSMOD
ENDATA
```

Figure 2: MPS basis file for trnsport.gms

```
  6  7
R0000001         COST
R0000002         SUPPLY     SEATTLE
R0000003         SUPPLY     SAN-DIEGO
R0000004         DEMAND     NEW-YORK
R0000005         DEMAND     CHICAGO
R0000006         DEMAND     TOPEKA
C0000001         X          SEATTLE     NEW-YORK
C0000002         X          SEATTLE     CHICAGO
C0000003         X          SEATTLE     TOPEKA
C0000004         X          SAN-DIEGO   NEW-YORK
C0000005         X          SAN-DIEGO   CHICAGO
C0000006         X          SAN-DIEGO   TOPEKA
C0000007         Z
```

Figure 3: GAMS to MPS mapping file for trnsport.gms

By default, the row and column names in the generated files correspond to the sequence number of the row or column and are eight characters wide. The MPS matrix and basis files correspond to the standard format. Note that the basis file is empty since the model has not been solved and the default solution values result in a unit basis. A unit basis means that all rows are basic and all variables are at bounds, the default format for the MPS basis file. We could have solved the model first and then use a second `solve` statement to write the MPS files. The MPS basis file will then contain entries for non-basic rows and basic columns. In the mapping file, the first line gives the number of rows and columns in the model. In the other lines the MPSWRITE generated row and column names is followed by the corresponding GAMS name (first the name of row/column followed by the indices). Since the model is being solved only with MPSWRITE, the listing file does not generate any solution and only reports that the solution is unknown.

Note also that the number of rows in the MPS file is one more than in the GAMS file. This is because GAMS creates an extra row containing only the objective variable. This transforms a problem in GAMS format, using an objective variable, into the MPS format, using an objective row. Finally, it is important to note that MPSWRITE always converts a problem into a minimization form when presenting the model in the MPS file format.

3 NAME GENERATION: A SHORT NOTE

The process involves compressing a name that can potentially be hundreds of characters long into a unique one of up to sixteen characters for the ANALYZE files and the mapping file and up to eight characters for the MPS and basis files. MPSWRITE allows for two ways of generating these names. The default names correspond to the sequence number of the particular row or column. However, these names, like the ones shown in Fig.1 are not always easy to identify with the corresponding GAMS entity without the aid of the mapping file. MPSWRITE, therefore, also allows for the more intelligent generation of names that correspond more closely with the original GAMS name and are therefore more easily identifiable. The procedure used for generating the names in this form is explained in detail in Section 9.

4 The Option File

The option file is called *mpswrite.opt*. This file is always read if it exists to avoid certain situations that occur due to adding a solver to the definition of MPSWRITE (as explained later in the chapter). In such a case, if say, BDMLP and MPSWRITE are called within the same solver step, the system does not know which option file to pick up. Therefore, the system always looks for *mpswrite.opt* when using MPSWRITE even if the name of the solver has been changed. The option file has the following format: each line contains either a comment (comments have a '*' as the first non-blank character) or an option. For example,

```
* write MPS names with a maximum width of 8 characters
namewid 8
* design name syntax based on GAMS name
nametyp 2
```

The contents of the option file are echoed to the screen and copied to the listing file.

5 ILLUSTRATIVE EXAMPLE REVISITED

Create the option file 'mpswrite.opt' and in it add the following line

```
nametyp 2
```

The option file is explained in greater detail in a following section. The MPS file generated on running the above problem is shown in Fig. 4.

```
NAME          GAMSMOD
* OBJECTIVE ROW HAS TO BE MINIMIZED
* ROWS       :        7
* COLUMNS    :        7
* NON-ZEROES :       20
ROWS
 N  +OBJ
 E  C
 L  SSE
 L  SSA
 G  DNE
 G  DCH
 G  DTO
```

```
COLUMNS
    XSENE     C         -0.225000000
    XSENE     SSE        1.000000000
    XSENE     DNE        1.000000000
    XSECH     C         -0.153000000
    XSECH     SSE        1.000000000
    XSECH     DCH        1.000000000
    XSETO     C         -0.162000000
    XSETO     SSE        1.000000000
    XSETO     DTO        1.000000000
    XSANE     C         -0.225000000
    XSANE     SSA        1.000000000
    XSANE     DNE        1.000000000
    XSACH     C         -0.162000000
    XSACH     SSA        1.000000000
    XSACH     DCH        1.000000000
    XSATO     C         -0.126000000
    XSATO     SSA        1.000000000
    XSATO     DTO        1.000000000
    Z         C          1.000000000
    Z         +OBJ       1.000000000


RHS
    RHS       SSE        350.0000000
    RHS       SSA        600.0000000
    RHS       DNE        325.0000000
    RHS       DCH        300.0000000
    RHS       DTO        275.0000000
BOUNDS
 FR BOUND     Z
ENDATA
```

Figure 4. MPS file for trnsport.gms with nametyp=2

Note that the names of the rows and columns are now more easily identifiable with the corresponding GAMS name. For example, the MPS variable `XSENE` corresponds to `x('Seattle','New-York')` and is much more easily identifiable than `C0000001` as in Fig.1. Note that in the newly generated name, the first character `X` represents the variable name `x`, the next two characters `SE` represent the first index label `Seattle` and the last two characters, `NE`, represent the second index label `New-York`.

6 USING ANALYZE WITH MPSWRITE

MPSWRITE uses two files to communicate with ANALYZE: The LP file and the system file. The LP file contains matrix information and the values of the rows and columns. ANALYZE then studies the linear program and its solution and interactively assists in diagnostic analysis. The syntax file is similar to the mapping file and serves to provide additional information for ANALYZE. Here, names of up to 16 characters are permitted to represent the GAMS variable or constraint.

In order to illustrate the generation of the ANALYZE LP and syntax files consider [TRNSPORT]. The options `domps` and `domap` are used to switch to ANALYZE format and are set to the value of 2. The new option file is shown below:

```
nametyp    2
domps      2
domap      2
```

The explanation of these options is given in greater detail in a following section. On solving [TRNSPORT] in the same form as in the previous section, the LP and syntax files are generated as *gams.lp* and *gams.syn* respectively and are shown in Figures 5 and 6.

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        8
OBJECTIV = +OBJ
ENDBASE


*** ROWS
C 7 0. 0.
B 0. 0.
SSE 3 -1.E20 350.
B 0. 0.
SSA 3 -1.E20 600.
B 0. 0.
DNE 2 325. 1.E20
B 0. 0.
DCH 2 300. 1.E20
B 0. 0.
DTO 2 275. 1.E20
B 0. 0.
+OBJ 1 -1.E20 1.E20
B 0. 0.


*** COLUMNS
XSENE 3 0. 1.E20
L 0. 0.
XSECH 3 0. 1.E20
L 0. 0.
XSETO 3 0. 1.E20
L 0. 0.XSANE 3 0. 1.E20


L 0. 0.
XSACH 3 0. 1.E20
L 0. 0.
XSATO 3 0. 1.E20
L 0. 0.
Z 2 -1.E20 1.E20
B 0. 0.
```

```
*** NONZEROS
C XSENE -0.22500000E+00
SSE XSENE 1.
DNE XSENE 1.
C XSECH -0.15300000E+00
SSE XSECH 1.
DCH XSECH 1.
C XSETO -0.16200000E+00
SSE XSETO 1.
DTO XSETO 1.
C XSANE -0.22500000E+00
SSA XSANE 1.
DNE XSANE 1.
C XSACH -0.16200000E+00
SSA XSACH 1.
DCH XSACH 1.
C XSATO -0.12600000E+00
SSA XSATO 1.
DTO XSATO 1.
C Z 1.
+OBJ Z 1.
```

Figure 5: ANALYZE LP file for 'trnsport.gms'

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
00   Universal Set
 SE SEATTLE
 SA SAN-DIEGO
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
I. canning plants
 SE SEATTLE
 SA SAN-DIEGO
J. markets
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
ROW SYNTAX
 C define objective function
 S observe supply limit at plant i &I.:2
 D satisfy demand at market j &J.:2
COLUMN SYNTAX
 X shipment quantities in cases &I.:2 &J.:4
 Z total transportation costs in thousands of dollars
ENDATA
```

Figure 6: ANALYZE Syntax file for 'trnsport.gms'

A further explanation on how to use these two files can be found in Section 10 and in greater detail in the ANALYZE manual.

7 The GAMS/MPSWRITE Options

There are 17 options that can be used to alter the structure and content of the various output files. They are classified by purpose and presented below.

7.1 Name Generation

`namewid`	integer	Maximum width of the names generated. NAMEWID can have a maximum width of 16. If any of the standard MPS files is being generated the maximum width is reduced to 8 (`domps=1, dobas=1`).
		(default = 8)
`nametyp`	integer	Type of names created for variables and equations.
		(default = 1)
	1	The generated names correspond to the ordinality number of the corresponding row or column
	2	More intelligent name generation base on GAMS names
`mcase`	integer	Mixed case name generation.
		(default=0)
	0	MPS names are all upper case
	1	MPS names use lower case
`Objlast`	Integer	Objective row last in MPS file.
		(default=0)
	0	Objective row first
	1	Objective row last

7.2 Name Design

Used if `nametyp=2`. A more detailed exposition on name generation can be found in the next section.

`labminwid`	integer	Minimum width for labels (set members) in new name
		(default = 1)
`labmaxwid`	integer	Maximum width for labels (set members) in new name. The upper bound for this is 4
		(default = 4)
`labperc`	real	100x is the percentage of clashes that are permitted in label names before renaming. In the name generation algorithm, if the percentage of clashes is more than `labperc`, then the width of the newly generated code for the label is increased.
		(default = 0.10)

`stemminwid`	integer	Minimum width of generated row or column. *(default = 1)*
`stemmaxwid`	integer	Maximum width of generated row or column name. The upper bound is 4. *(default = 4)*
`stemperc`	real	100x is the percentage of clashes permitted in row or column names before renaming (similarly to `labperc`). *(default = 0.10)*
`textopt`	integer	Option to process special @ symbol in GAMS symbol text during the generation of ANALYZE syntax files. *(default = 0)*
	0	Ignore @ in the GAMS symbol text.
	1	ANALYZE specific option related to the *gams.syn* file. The macro symbol @ in the GAMS symbol text is recognized and processed accordingly.
`fixed`	integer	Option for fields in LP file. This is again an ANALYZE specific option. *(default = 1)*
	0	Fixed fields for writing LP file.
	1	The LP files is squeezed to remove extra spaces

7.3 Using Files

`usesol`	integer	Option to control the use of primal and dual information. *(default= 0)*
	0	Use matrix file for obtaining dual information. This is then used for the generation of the basis file and the ANALYZE LP file. MPSWRITE will also generate an empty solution file on completion to pass return information required by GAMS.
	1	Use the GAMS solution file to obtain primal and dual solution values. If the model contains nonlinear terms, the partial derivatives (matrix coefficients) and the right-hand-side are also updated to reflect the linearization around the new point. If the matrix is written in ANALYZE LP format, the solution status will be the one from the solution file.

7.4 Writing Files

`domps`	integer	Option to output of the problem matrix. *(default= 1)*
	0	No matrix file is written
	1	MPS file written as *gams.mps*
	2	Matrix file in ANALYZE LP format is written as *gams.lp*

`btol`	real	Basis tolerance when generating the ANALYZE file *`gams.lp`*. Optimization involves calculations with floating point numbers and is inherently noisy. MPSWRITE re-computes the row activity levels for linear and nonlinear models and the basis tolerance `btol` is used to determine the final basis status. In order to avoid spurious warnings in subsequent steps like ANALYZE, activity levels are reset to the value of the appropriate bound when within the tolerance level. *(default = 1.0d-8)*
`pinf`	real	The value to be use for plus infinity when writing an ANALYZE LP file. *(default = 1.0d20)*
`dobas`	integer	controls the output of a file containing basis information *(default = 1)*
	0	No basis file is written
	1	Basis file is written as *`gams.bas`*. As explained earlier, the basis file can be written based on primal and dual information from the solution file if `usesol=1` or else the information is obtained from the matrix file. Potential complications arise with superbasic variables in nonlinear problems. Although most modern LP codes allow work with a similar concept, the standard format of the MPS basis file does not allow for this situation. The OSL specific 'BS' format is used to indicate superbasic activity.
`domap`	integer	controls the output of a file containing name mapping information. *(default = 1)*
	0	no mapping file is written
	1	mapping file containing generated name followed by GAMS name is written in *`gams.map`*.
	2	syntax file with ANALYZE format is written in *`gams.syn`*. Superbasic activities cannot be represented with the ANALYZE LP format. When such activities are encountered, one of the bounds is reset to the activity level, the activity is made non-basic, and a comment is inserted with an appropriate message and the value of the original bound. The lower or upper bound is selected to make the modified problem optimal (according to the sign of the marginal).

Therefore, by default, the MPS, basis and mapping files are generated as *`gams.mps`*, *`gams.bas`*, and *`gams.map`* respectively. The dual information for the generation of the basis file is read from the GAMS matrix file. The generated names have a default width of 8. The effect of various options on the output files is demonstrated through [TRNSPORT] in Section 9.

8 Name Generation Examples

In GAMS, the row and column names have an identifier followed by up to 10 indices which are also identifiers. An example of a name from [TRNSPORT] is `x(Seattle,New-York)`. As a result of this, the total length of a single row or column name can be hundreds of characters long.

In ANALYZE, the names can be up to 16 characters. Character positions determine the meaning, for instance `XSANE` may mean shipping (`X`) from San-Diego (`SA`) to New-York (`NE`). The ANALYZE command 'explain' shows you the meaning of a name in this way based on the information in the syntax file. In MPS files, the names are shorter and have to be less than 8 characters.

The way these names are generated from the GAMS name is as follows: Stems are generated for the equation/variable names. The width of these stems is within the limits provided by the users through the option file. `stemminwid` and `stemmaxwid` provide the minimum and maximum width for the rows and column names while `labminwid` and `labmaxwid` provide similar limits for the label names. The procedure for the generation of these stems is given below.

Initially, the first few characters of the name are chosen to form the name stem or Label name, the width corresponding to the minimum width permitted through the option file. If all the stems that are created in this manner are unique, then the procedure is complete and the stem names created are used. If there are repetitions or non-uniqueness in the stems created then the fraction of these repetitions (the number of repetitions divided by the number of names) is checked against the maximum permissible fraction allowed through the option file through the setting of `labperc` and `stemperc`. If the fraction is above the permissible limit, then the size of the stem or label is increased by one until the maximum width is reached or the fraction is less than the permissible limit. When this is achieved, the remaining clashes are resolved by leaving the first occurrence of the stem or label name as it is and altering the other occurrences of the stem or label name. These alterations are done by first changing the last character of the stem to a unique one. If this is not possible, the procedure attempts to change the second to the last character to make the stem or label unique. If one cannot still find a unique name, progressively the more significant positions of the stem or label are changed until a unique name is found. Only alphanumeric characters are considered during the generation process.

In order to illustrate the procedure explained above, consider the following set of label names

```
NEW-YORK, BOSTON, PITTSBURGH, NEWARK, WASHINGTON, BUFFALO
```

Let the values of labminwid, labmaxwid and labperc be 1,3 and 0.1 respectively. Initially stems of width 1 corresponding to labminwid are created: N, B, P, N, W, B. The result is that 0.33 of the number of names are repeated (2 repetition of a total of 6 names). This is greater than the permitted maximum of 0.1. The width of the label is therefore increased to two, creating the labels, NE, BO, PI, NE, WA, BU. This now increases the number of unique label names but still leaves the fraction of repetitions at 0.17 and so the label width has to be further increased by one to 3. The resulting label are NEW, BOS, PIT, NEW, WAS, BUF. The fraction of repetitions is unchanged, but a further increase in the label width is not possible because the maximum allowable width (labmaxwid) has been reached. The remaining non-unique label names have to be altered. The first occurrence of NEW is left unchanged while the second occurrence is altered to NEA, which is a unique name. This leads to the following set of unique label names that are finally obtained:

```
NEW, BOS, PIT, NEA, WAS, BUF
```

In this manner, unique label names are generated for the label names and stem names for the equation/variable names separately. Once this is done, the name associated with any particular row or column is created by putting together the stem name of the equation/variable and the label of all the indices associated with it in order. For example in the variable x(Seattle, New-York) in [TRNSPORT], if the stem name of the variable is X and the stem names of the associated labels are SE and NE, the name of the corresponding column is XSENE.

However, there is a limit of namewid characters that the row/column name can have. If the width of the labels or equation/variable stems is too large or if the row/column has too many indices, the size of the row or column created in the manner described above may lead to a width greater than namewid. For example, consider that namewid is 16 and that the width of the stems of the labels and equation/variable names created by the procedure described above is 3. Now consider the variable y(i,j,k,l,m) which has 5 index positions. The width of any instance of the variable would be 3 + 5(3) = 18. This is clearly above the limit set by the option file. In such a situation, the latter indices are collapsed to form new index sets and labels so as to limit the name width to a maximum of namewid. In the example above, the index sets l and m are collapsed to create a new set and new unique labels for each occurring combination of labels of l and m in y(i,j,k,l,m). The new label names are generated by simply using the ordinality of the row or column.

The effect of these procedures is shown in the examples below.

Example 3: Consider solving trnsport.gms with the following entries in the option file

```
nametyp    2
namewid    8
labminwid  1
labmaxwid  3
labperc    0.25
stemminwid        2
stemmaxwid        3
stemperc   0.25
```

The resulting GAMS to MPS mapping file that explains the new names is shown in Figure 7. Notice here that due to the increase in the allowable percentage of clashes in the stems of the label names and the equation/variable names, stems with just one character were sufficient for the label names. A `stemminwid` of 2 forced the stems of the equation/variable names to have two characters. It must be pointed out that all extra positions are padded with the '.' character. So the variables X and Z being smaller than `stemminwid` were changed to `X.` and `Z.` . `X.SN` therefore refers to the GAMS column `X(Seattle, New-York)`.

```
  6  7
CO              COST
SUS             SUPPLY     SEATTLE
SUA             SUPPLY     SAN-DIEGO
DEN             DEMAND     NEW-YORK
DEC             DEMAND     CHICAGO
DET             DEMAND     TOPEKA
X.SN            X          SEATTLE    NEW-YORK
X.SC            X          SEATTLE    CHICAGO
X.ST            X          SEATTLE    TOPEKA
X.AN            X          SAN-DIEGO  NEW-YORK
X.AC            X          SAN-DIEGO  CHICAGO
X.AT            X          SAN-DIEGO  TOPEKA
Z.              Z
```

Figure 7: GAMS to MPS Mapping file with `LABPERC, STEMPERC = 0.25`

Example 4: In the option file described above, the width of the permissible name is set to 3 by `namewid 3`. Also, to create ANALYZE files, add `domap 2` and `domps 2`. Since the equation/variable name stems still had to have at least two characters because of `stemminwid`, this example is used to show the effect of the collapsing process on the names generated. The variable `x(i,j)` requires the collapsing of its indices since two indices would not be able to fit into the last remaining position. The resulting ANALYZE syntax file (*`gams.syn`*) that explains the new labels and set that were created due to the collapsing is shown in Figure 8. The new set (called 3) consists of the 6 members of `x(i,j)` and has labels whose names depend on the ordinality of the occurrence of label combinations in (I,J). The new row/column names that are generated can be seen in the ANALYZE LP file (*`gams.lp`*) shown in Figure 9. Notice now that a 3 character name `X.1` represent `x(Seattle, New-York).` This example may seem rather forced due to the relatively small number of indices in the equations and variables, however it should be realized that when the number of indices becomes larger, this process is inevitable.

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
0    Universal Set
 S SEATTLE
 A SAN-DIEGO
 N NEW-YORK
 C CHICAGO
 T TOPEKA
I canning plants
 S SEATTLE
 A SAN-DIEGO
J markets
 N NEW-YORK
 C CHICAGO
 T TOPEKA
3 NEW SET FOR X
 1 SEATTLE   .NEW-YORK
 2 SEATTLE   .CHICAGO
 3 SEATTLE   .TOPEKA
 4 SAN-DIEGO .NEW-YORK
 5 SAN-DIEGO .CHICAGO
 6 SAN-DIEGO .TOPEKA
ROW SYNTAX
 CO define objective function
 SU observe supply limit at plant i &I:3
 DE satisfy demand at market j &J:3
COLUMN SYNTAX
 X. shipment quantities in cases &3:3
 Z. total transportation costs in thousands of dollars
ENDATA
```

Figure 8: ANALYZE Syntax file with NAMEWID=3

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        3
OBJECTIV = +OBJ
ENDBASE


*** ROWS
CO 7 0. 0.
B 0. 0.
SUS 3 -1.E20 350.
B 0. 0.
SUA 3 -1.E20 600.
B 0. 0.
DEN 2 325. 1.E20
B 0. 0.
DEC 2 300. 1.E20
B 0. 0.
DET 2 275. 1.E20
B 0. 0.
+OBJ 1 -1.E20 1.E20
B 0. 0.
```

```
*** COLUMNS
X.1 3 0. 1.E20
L 0. 0.
X.2 3 0. 1.E20
L 0. 0.
X.3 3 0. 1.E20
L 0. 0.
X.4 3 0. 1.E20
L 0. 0.
X.5 3 0. 1.E20
L 0. 0.
X.6 3 0. 1.E20
L 0. 0.
Z. 2 -1.E20 1.E20
B 0. 0.


*** NONZEROS
CO X.1 -0.22500000E+00
SUS X.1 1.
DEN X.1 1.
CO X.2 -0.15300000E+00
SUS X.2 1.
DEC X.2 1.
CO X.3 -0.16200000E+00
SUS X.3 1.
DET X.3 1.
CO X.4 -0.22500000E+00
SUA X.4 1.
DEN X.4 1.
CO X.5 -0.16200000E+00
SUA X.5 1.
DEC X.5 1.
CO X.6 -0.12600000E+00
SUA X.6 1.
DET X.6 1.
CO Z. 1.
+OBJ Z. 1.
```

Figure 9: ANALYZE LP file with NAMEWID=3

```
$TITLE  A TRANSPORTATION PROBLEM (TRNSPORT,SEQ=1)
$OFFUPPER

  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 from @i to @j
     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/ ;

OPTION LP = MPSWRITE ;

SOLVE TRANSPORT USING LP MINIMIZING Z ;

DISPLAY X.L, X.M ;
```

Figure 10: Modified 'trnsport.gms' for Example (A.3)

```
* SYNTAX FILE GENERATED BY GAMS/MPSWRITE
00   Universal Set
 SE SEATTLE
 SA SAN-DIEGO
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
I. canning plants
 SE SEATTLE
 SA SAN-DIEGO
J. markets
 NE NEW-YORK
 CH CHICAGO
 TO TOPEKA
ROW SYNTAX
 C define objective function
 S observe supply limit at plant &I.:2
 D satisfy demand at market &J.:2
COLUMN SYNTAX
 X shipment quantities in cases from &I.:2 to &J.:4
 Z total transportation costs in thousands of dollars
ENDATA
```

Figure 11: ANALYZE Syntax file with TEXTOPT=1

```
* file created by gams
* BASE
OPTIMAL  = MINIMIZE
STATUS   = UNKNOWN
NAME     = GAMS
ROWS     =        7
COLUMNS  =        7
NONZEROS =       20
LENGTH   =        8
OBJECTIV = +OBJ
ENDBASE


*** ROWS
C                     7    0.00000000E+00    0.00000000E+00
B                          0.00000000E+00    0.00000000E+00
SSE                   3   -0.10000000E+21    0.35000000E+03
B                          0.00000000E+00    0.00000000E+00
SSA                   3   -0.10000000E+21    0.60000000E+03
B                          0.00000000E+00    0.00000000E+00
DNE                   2    0.32500000E+03    0.10000000E+21
B                          0.00000000E+00    0.00000000E+00
DCH                   2    0.30000000E+03    0.10000000E+21
B                          0.00000000E+00    0.00000000E+00
DTO                   2    0.27500000E+03    0.10000000E+21
B                          0.00000000E+00    0.00000000E+00
+OBJ                  1   -0.10000000E+21    0.10000000E+21
B                          0.00000000E+00    0.00000000E+00


*** COLUMNS
XSENE                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
XSECH                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
XSETO                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
XSANE                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
XSACH                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
XSATO                 3    0.00000000E+00    0.10000000E+21
L                          0.00000000E+00    0.00000000E+00
Z                     2   -0.10000000E+21    0.10000000E+21
B                          0.00000000E+00    0.00000000E+00


*** NONZEROS
C                 XSENE              -0.22500000E+00
SSE               XSENE               0.10000000E+01
DNE               XSENE               0.10000000E+01
C                 XSECH              -0.15300000E+00
SSE               XSECH               0.10000000E+01
DCH               XSECH               0.10000000E+01
C                 XSETO              -0.16200000E+00
SSE               XSETO               0.10000000E+01
DTO               XSETO               0.10000000E+01
C                 XSANE              -0.22500000E+00
SSA               XSANE               0.10000000E+01
DNE               XSANE               0.10000000E+01
C                 XSACH              -0.16200000E+00
SSA               XSACH               0.10000000E+01
DCH               XSACH               0.10000000E+01
C                 XSATO              -0.12600000E+00
SSA               XSATO               0.10000000E+01
DTO               XSATO               0.10000000E+01
C                 Z                   0.10000000E+01
+OBJ              Z                   0.10000000E+01
```

Figure 12: ANALYZE LP file with FIXED=1

Example 5: This example is used to demonstrate the effect of the options `fixed` and `textopt`. Consider the same gams input file *`trnsport.gms`* with the following modification: replace the text following the definition of the variable `x(i,j)` by '`shipment quantities in cases from @i to @j`' and that following the definitions of equations `supply` and `demand` by '`observe supply at plant @i`' and 'satisfy demand at market @j' respectively. The resulting gams file is shown in Figure 10. The macro character `@` is used as a flag for GAMS/MPSWRITE to recognize the following characters as a set name and replace them by a string recognizable by ANALYZE. Now include the option file shown below.

```
nametyp          2
dobas            0
domap            2
domps            2
textopt          1
fixed            1
```

The options, `textopt 1` and `fixed 1` have been added relative to the previous examples. The resulting ANALYZE syntax file is shown in Figure 11. Notice the string that replaces the macros `@i` and `@j`. In the string, ANALYZE recognizes the characters between '`&`' and '`:`' as representing the stem of the index name corresponding to the string following the '`@`' macro in the gams file, and the number immediately following the ':' as the number of characters from the left of the row/column name that the particular set label stem begins. As an illustration, in the row syntax for `X`, we see the strings `&I.:2` and `&J.:4`. This means that in any row of `X`, say `XSENE`, the label corresponding to `I` begins at the second position (`SE`) while the label corresponding to `J` begins at the fourth position (`NE`). This is used by the ANALYZE command EXPLAIN. For further information on this topic, consult the ANALYZE manual.

The ANALYZE LP file shown in Figure 11 demonstrates the effect of the option `fixed` which uses a fixed field format for the LP file. This makes the file size larger but at the same time is easier to read.

9 INTEGRATING GAMS AND ANALYZE

In this section a number of examples are given to demonstrate the use of ANALYZE with GAMS. This description is intended for users already familiar with ANALYZE. For more information on ANALYZE you should consult the User's Guide for ANALYZE: Harvey J. Greenberg, A Computer-Assisted Analaysis System for Mathematical Programming Models and Solutions: A Users's Guide for ANALYZE, Kluwer Academic Publishers, Boston, 1993.

GAMS takes advantage of a special ANALYZE feature which allows 16 character long names instead of the usual 8 character MPS type names. In order to use this feature, the problem has to presented in an ANALYZE specific format, the ANALYZE LP file, which needs further conversion into an ANALYZE PCK file format before it can be read by ANALYZE. The ANALYZE Utility LPRENAME is needed to rewrite the GAMS generated LP file into an ANALYZE PCK file

The general steps to translate an LP file are:

```
>LPRENAME
 < LPRENAME credits etc >
 LPRENAME... LOAD GAMS
 LPRENAME... WRITE GAMS
 LPRENAME... QUIT
```

The command 'LOAD GAMS' instructs LPRENAME to read the input file GAMS.LP. The command `write` will generate a file called *`gams.pck`* which is a binary (packed) version of *`gams.lp`*. For the purposes of this document, this is all you needs to know about the LPRENAME function, further details can be found in Chapter 8 of the ANALYZE User's Guide.

There are various ways how you can generate the appropriate input files for ANALYZE and different methods of invoking the analysis software. The choice will depend on the kind of analysis is required and how frequent the GAMS-ANALYZE link will be used. Several examples will be used to illustrate different GAMS/ANALYZE set-ups for DOS or Windows type environments.

Example 6: This example shows how to generate the ANALYZE LP and SYN files and save them for further processing. An MPSWRITE option file will be used to specify the types of files required and name design option selected. The 'mpswrite.opt' file will contain:

```
* design meaningful names
nametyp    2
* analyze can use wide names
namewid    16
* write an analyze syntax file
domap      2
* write an analyze matrix file
domps      2
* don't write basis file
dobas      0
```

In the GAMS file a solve statement is used to call the standard MPSWRITE subsystem. Note that MPSWRITE is invoked after a `solve` statement in order to pass correct dual information. Once again [TRNSPORT] is used for illustration. The following slice of code is added at the end of the file:

```
solve transport minimizing z using lp;
display x.l, x.m;
* additions follow below
option lp=mpswrite;
solve transport minimizing z using lp
```

After running the modified TRNSPORT model you may want to save the file *gams.lp* and *gams.syn* under a different name. For example, you could just copy and translate the files as follows:

```
>GAMS TRNSPORT
    < GAMS log >
>COPY GAMS.LP  TRNSPORT.LP
>COPY GAMS.SYN TRNSPORT.SYN
>LPRENAME
  < translate .lp file into a .pck file
  .. LOAD TRNSPORT
  .. WRITE TRNSPORT
  .. QUIT
>ANALYZE
  < various credits etc.>
  ANALYZE... READIN PAC TRNSPORT
  ANALYZE... READIN SYN TRNSPORT
  < now we are ready to use ANALYZE commands >
  ANALYZE... QUIT
```

Example 7: This example shows how you can partially automate the name generation, MPS renaming and loading of the model into ANALYZE.

```
@echo off
:
echo * MPSWRITE Option File > mpswrite.opt
echo nametyp  2            >> mpswrite.opt
echo namewid 16            >> mpswrite.opt
echo dobas    0            >> mpswrite.opt
echo domap    2            >> mpswrite.opt
echo domps    2            >> mpswrite.opt
echo textopt  1            >> mpswrite.opt
gams trnsport lp=mpswrite
:
echo LOAD  GAMS  > profile.exc
echo WRITE GAMS >> profile.exc
echo QUIT       >> profile.exc
lprename
:
echo $ ============   > profile.exc
echo $ GAMS/ANALYZE  >> profile.exc
echo $ ============  >> profile.exc
echo READIN PAC GAMS >> profile.exc
echo READIN SYN GAMS >> profile.exc
echo SUMMARY         >> profile.exc
echo $               >> profile.exc
echo $  SUBMAT ROW * >> profile.exc
echo $  SUBMAT COL * >> profile.exc
echo $  PICTURE      >> profile.exc
echo $               >> profile.exc
echo $  HELP         >> profile.exc
echo $               >> profile.exc
```

```
if not exist credits.doc goto ok
: need to remove credits to prevent early prompt
if exist credits.hld delete credits.hld
rename credits.doc credits.hld
:ok
analyze
```

The option and profile files for GAMS, LPRENAME and ANALYZE are created on the fly. The exact content of these files will depend on the version and installation of ANALYZE.

Example 8: This example shows how you can ANALYZE as a GAMS solver . You need to consult your GAMS and ANALYZE system administrator to install the scripts with the right name and in the proper place. For example, when running under Windows 95 we could add a solver called ANALYZE that uses the same settings as MPSWRITE in the appropriate GAMS configuration file. The new ANALYZE script then could look as follows.

```
@echo off
: gmsan_95.bat: Command Line Interface for Win 95
echo * ANALYZE defaults     > analyze.opt
echo nametyp  2            >> analyze.opt
echo namewid 16            >> analyze.opt
echo dobas    0            >> analyze.opt
echo domap    2            >> analyze.opt
echo domps    2            >> analyze.opt
echo textopt  1            >> analyze.opt
echo * MPSWRITE Option     >> analyze.opt
: append mpswrite options
if exist mpswrite.opt copy analyze.opt + mpswrite.opt analyze.opt
gmsmp_nx.exe    %4
if exist analyze.opt del analyze.opt
:
echo LOAD  GAMS  > profile.exc
echo WRITE GAMS >> profile.exc
echo QUIT       >> profile.exc
:lprename > lprename.out
LPRENAME
:
echo $ ============    > profile.exc
echo $ GAMS/ANALYZE   >> profile.exc
echo $ ============   >> profile.exc
echo READIN PAC GAMS  >> profile.exc
echo READIN SYN GAMS  >> profile.exc
echo SUMMARY          >> profile.exc
echo $                >> profile.exc
echo RETURN           >> profile.exc
if not exist credits.doc goto ok
: need to remove credits to prevent early prompt
if exist credits.hld delete credits.hld
rename credits.doc credits.hld
:ok
ANALYZE
```

Note that LPRENAME and ANALYZE are assumed to be accessible through the path or can be found in the current directory. If this is not the case, you has to enter the appropriate path to the script file above. Also note the use of the ANALYZE profile feature. The *profile.exc* file is executed only after the credits have been processed. The only way to avoid having to respond with Y/N to ANALYZE questions interactively is the renaming of the *credits.doc* file. If no *credits.doc* file is found by ANALYZE, it will immediately look for a *profile.exc* file and execute the commands included in this file.

Let us return to the [TRNSPORT] example and call ANALYZE after we have computed an optimal solution as follows:

```
solve transport using lp minimizing cost;
option lp=analayze;
solve transport using lp minmizing cost;
```

The second solve statement will generate the appropriate input files for ANALYZE and load the problem into the analyze workspace. Not that ANALYZE has recognized that the model is optimal and the model is ready to be inspected using ANALYZE commands as shown below:

```
                          ANALYZE    Version 10.0
                                      by
                             Harvey J. Greenberg

 PLEASE LOCK CAPS

  ============
 GAMS/ANALYZE
  ============
 Packed LP read from GAMS.PCK

 Syntax read from GAMS.SYN

 Problem GAMS is to MINIMIZE +OBJ.
 RHS=none  RANGE=none  BOUND=none
 Source file: GAMS.PCK
 Synatx file: GAMS.SYN
 Solution: GAMS    Solution Status: OPTIMAL
 LP Statistics:
     7 Rows (1 free) and 7 Columns (1 fixed)
     20 Nonzeroes (11 distinct... 14 ones).
     Density = 40.8163%  Entropy = 55%  Unity = 70%
 Row    C     has the most nonzeroes = 7.
 Column XSENE has the most nonzeroes = 3.

 RETURN...back to terminal
ANALYZE ...
```

Using the flexibility of both systems, GAMS and ANALYZE, you can further enhance the above examples and provide application tailored analytic capabilities for standard GAMS applications.

GAMS/OSL

1 INTRODUCTION

This document describes the GAMS interface to OSL.

OSL is the IBM Optimization Subroutine Library, containing high performance solvers for LP (linear programming), MIP (mixed integer programming) and QP (quadratic programming) problems. GAMS does not support the QP capabilities of OSL, you have to use a general non-linear solver like MINOS or CONOPT for that.

OSL offers quite a few algorithms and tuning parameters. Most of these are accessible by the GAMS user through an option file. In most cases GAMS/OSL should perform satisfactory without using any options.

2 HOW TO RUN A MODEL WITH OSL

OSL is capable of solve models of the following types: LP, RMIP and MIP. If you did not specify OSL as a default LP, RMIP or MIP solver, then you can use the following statement in your GAMS model:

```
option lp=osl;    { or RMIP or MIP }
```

It should appear before the SOLVE statement.

3 OVERVIEW OF OSL

OSL offers several algorithms for solving LP problems: a primal simplex method (the default), a dual simplex method, and three interior point methods: primal, primal-dual and primal-dual predictor-corrector. For network problems there is a network solver.

Normally the primal simplex method is a good method to start with. The simplex method is a very robust method, and in most cases you should get good performance with this solver. For large models that have to be solved many times it may be worthwhile to see if one of the other methods gives better results. Also changing the tuning parameters may influence the performance. The `method` option can be used to use another algorithm than the default one.

3.1 The Simplex Method

The most used method is the primal simplex method. It is very fast, and allows for restarts from an advanced basis. In case the GAMS model has multiple solves, and there are relatively minor changes between those LP models, then the solves after the first one will use basis information from the previous solve to do a 'jump start'. This is completely automated by GAMS and normally you should not have to worry about this.

In case of a 'cold start' (the first solve) you will see on the screen the message 'Crash...'. This will try to create a better basis than the scratch ('all-slack') basis the Simplex method would normally get. The crash routine does not use much time, so it is often beneficial to crash. Crashing is usually not used for subsequent solves because that would destroy the advanced basis. The default rule is to crash when GAMS does not pass on a basis, and not to crash otherwise. Notice that in a GAMS model you can use the `bratio` option to influence GAMS whether or not to provide the solver with a basis. The default behavior can be changed by the `crash` option in the option file.

By default the model is also scaled. Scaling is most of the time beneficial. It can prevent the algorithm from breaking down when the matrix elements have a wide range: i.e. elements with a value of 1.0e-6 and also of 1.0e+6. It can also reduce the solution time.

The presolver is called to try to reduce the size of the model. There are some simple reductions that can be applied before the model is solved, like replacing singleton constraints (i.e. `x =l= 5 ;`) by bounds (if there was already a tighter bound on X we just can remove this equation). Although most modelers will already use the gams facilities to specify bounds (`x.lo` and `x.up`), in many cases there are still possibilities to do these reductions. In addition to these reductions OSL can also remove some redundant rows, and substitute out certain equations. The presolver has several options which can be set through the `presolve` option.

The presolve may destroy an advanced basis. Sometimes this will result in very expensive restarts. As a default, the presolve is not used if an advanced basis is available. If using the presolve procedure is more useful than the use of an advanced basis, one can still force a presolve by using an option file.

GAMS/OSL uses the order: scale, presolve, crash. There is no possibility to change this order unless you have access to the source of the GAMS/OSL program.

After the model is solved we have to call the postsolver in case the presolver was used. The postsolver will reintroduce the variables and equations the presolve substituted out, and will calculate their values. This solution is an optimal solution, but not a basic solution. By default we call simplex again to find an optimal basis. This allows us to restart from this solution. It is possible to turn off this last step by the option `postsolve 0`.

Occasionally you may want to use the dual simplex method (for instance when the model is highly primal degenerate, and not dual degenerate, or when you have many more rows than columns). You can use the `method dsimplex` option to achieve this. In general the primal simplex (the default) is more appropriate: most models do not have the characteristics above, and the OSL primal simplex algorithm is numerically more stable.

The other options for the Simplex method like the refactorization frequency, the Devex option, the primal and the dual weight and the change weight option are only to be changed in exceptional cases.

3.2 The interior point methods

OSL also provides you with three interior point solvers. You should try them out if your model is large and if it is not a restart. The primal-dual barrier method with predictor-corrector is in general the best algorithm. This can be set by `method interior3`. Note that the memory estimate of OSL is in many cases not sufficient to solve a model with this method. You can override OSL's estimate by adding to using the workspace model suffix as shown in Section 4.2. We have seen models where we had to ask for more than twice as much as the estimate. It is worthwhile to check how the interior points are doing on your model especially when your model is very large.

3.3 The network method

A linear model can be reformulated to a network model if

- The objective variable is a free variable,
- The objective variable only appears in the objective function, and not somewhere else in the model. This in fact defines the objective function.
- The objective function is of the form `=e=.`
- Each variable appears twice in the matrix (that is excluding the objective function) once with a coefficient of +1 and once with a coefficient of -1. In case there is a column with two entries that are the same, GAMS/OSL will try a row scaling. If there are no matrix entries left for a column (only in the objective function) or there is only one entry, GAMS/OSL will try to deal with this by adding extra rows to the model.

4 GAMS OPTIONS

The following GAMS options are used by GAMS/OSL.

4.1 Options specified through the Option statement

The following options are specified through the option statement. For example,

```
set iterlim = 100 ;
```

sets the iterations limit to 100.

`iterlim`	Sets the iteration limit. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution.
`reslim`	Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution.
`optca`	Absolute optimality criterion for a MIP problem.
`optcr`	Relative optimality criterion for a MIP problem.
`bratio`	Determines whether or not to use an advanced basis.
`sysout`	Will echo the OSL messages to the GAMS listing file. This option is useful to find out exactly what OSL is doing.

4.2 Options specified through model suffixes

The following options are specified through the use of model suffix. For example,

```
mymodel.workspace = 10 ;
```

sets the amount of memory used to 10 MB. `Mymodel` is the name of the model as specified by the `model` statement. In order to be effective, the assignment of the model suffix should be made between the `model` and `solve` statements.

`workspace`	Gives OSL x MB of workspace. Overrides the memory estimation.
`optfile`	Instructs OSL to read the option file *osl.opt*.

`cheat`	Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OPTCA option). The OSL option *improve* overrides the GAMS cheat parameter.
`cutoff`	Cutoff value. When the Branch and Bound search starts, the parts of the tree with an objective worse than x are deleted. Can speed up the initial phase in the Branch and Bound.
`prioropt`	Instructs OSL to use priority branching information passed by GAMS through the `variable.prior` parameters.

5 SUMMARY OF OSL OPTIONS

All these options should be entered in the option file *osl.opt* after setting the `mymodel.optfile` parameter to 1. The option file is not case sensitive and the keywords must be given in full. Examples for using the option file can be found at the end of this section.

5.1 LP algorithmic options

`crash`	crash an initial basis
`iterlim`	iteration limit
`method`	solution method
`pdgaptol`	barrier method primal-dual gap tolerance
`presolve`	perform presolve. Reduce size of model.
`reslim`	resource limit
`simopt`	simplex option
`scale`	perform scaling
`toldinf`	dual feasibility tolerance
`tolpinf`	primal feasibility tolerance
`workspace`	memory allocation

5.2 MIP algorithmic options

`bbpreproc`	branch and bound preprocessing
`cutoff`	cutoff or incumbent value
`cuts`	allocate space for extra cuts
`degscale`	scale factor for degradation
`improve`	required level of improvement over current best integer solution
`incore`	keep tree in core
`iweight`	weight for each integer infeasibility
`maxnodes`	maximum number of nodes allowed
`maxsols`	maximum number of integer solutions allowed
`optca`	absolute optimality criterion
`optcr`	relative optimality criterion
`strategy`	MIP strategy
`target`	target value for objective
`threshold`	supernode processing threshold
`tolint`	integer tolerance

5.3 Screen and output file options

`bastype`	format of basis file
`creatbas`	create basis file
`creatmps`	create MPS file
`iterlog`	iteration log
`mpstype`	type of MPS file
`networklog`	network log
`nodelog`	MIP log

5.4 Advanced LP Options

`adjac`	save storage on AA^T
`chabstol`	absolute pivot tolerance for Cholesky factorization
`chweight`	rate of change of multiplier in composite objective function
`chtinytol`	cut-off tolerance in Cholesky factorization
`crossover`	when to switch to simplex
`densecol`	dense column threshold
`densethr`	density threshold in Cholesky
`devex`	devex pricing method
`droprowct`	constraint dropping threshold
`dweight`	proportion of feasible objective used in dual infeasible solution
`factor`	refactorization frequency
`fastits`	switching frequency to devex
`fixvar1`	tolerance for fixing variables in barrier method when infeasible
`fixvar2`	tolerance for fixing variables in barrier method when feasible
`formntype`	formation of normal matrix
`maxprojns`	maximum number of null space projections
`muinit`	initial value for μ
`mulimit`	lower limit for μ
`mulinfac`	multiple of μ to add to the linear objective
`mufactor`	reduction factor for μ in primal barrier method
`nullcheck`	null space checking switch
`objweight`	weight given to true objective function in phase I of primal barrier
`pdstepmult`	step-length multiplier for primal-dual barrier
`pertdiag`	diagonal perturbation in Cholesky factorization

`possbasis`	potential basis flag
`postsolve`	basis solution required
`projtol pweight`	projection error tolerance starting value for multiplier in composite objective function
`rgfactor`	reduced gradient target reduction
`rglimit`	reduced gradient limit
`simopt`	simplex option
`stepmult`	step-length multiplier for primal barrier algorithm

5.5 Examples of GAMS/OSL option file

The following option file *osl.opt* may be used to force OSL to perform branch and bound preprocessing, a maximum of 4 integer solutions and to provide a log of the branch and bound search at every node.

```
bbpreproc 1
maxsols 4
nodelog 1
```

6 DETAILED DESCRIPTION OF OSL OPTIONS

`adjac`	integer		Generation of AA^T *(default = 1)*
		0	Save storage on forming AA^T in the interior point methods.
		1	Use a fast way of computing AA^T.
`bastype`	integer		Format of basis file *(default = 0)*
		0	do not write level values to the basis file
		1	write level values to the basis file
`bbpreproc`	integer		Preprocess the Branch and Bound tree *(default = 0)*.
		0	Do not preprocess the 0-1 structure.
		1	Use super-nodes, copy matrix
		2	Regular branch-and-bound, copy matrix
		3	Use super-nodes, overwrite matrix
		4	Regular branch-and-bound, overwrite matrix

Note: The pre-processor examines only the 0-1 structure. On models with only general integer1 variables (i.e. integer variables with other bounds than 0 and 1) the preprocessor will not do any good. A message is written if this happens.
The preprocessor needs space for extra cuts. If no space available, the branch-and-bound search may fail. Use the `cuts` option to specify how much extra room has to be allocated for additional cuts. Notice that the `presolve` already may reduce the size of the model, and so create extra space for additional cuts.

`chabstol`	real	Absolute pivot tolerance for the Cholesky factorization. Range - [1.0e-30, 1.0e-6] *(default = 1.0e-15)*
`chtinytol`	real	Cut-off tolerance in the Cholesky factorization. Range - [1.0e-30, 1.0e-6] *(default = 1.0e-18)*
`chweight`	real	Rate of change for multiplier in composite objective function. Range - [1e-12,1] *(default = 0.5)*

Note: This value determines the rate of change for `pweight` or `dweight`. It is a nonlinear factor based on case-dependent heuristics. The default of 0.5 gives a reasonable change if progress towards feasibility is slow. A value of 1.0 would give a greater change, while 0.1 would give a smaller change, and 0.01 would give very slow change.

`Crash`	integer		Crash an initial basis. This option should not be used with a network or interior point solver. The option is ignored if GAMS provides a basis. To tell GAMS never to provide the solver with a basis `use 'option bratio = 1;'` in the GAMS program. *(default = 1 if no basis provided by GAMS)*
		0	No crash is performed
		1	Dual feasibility may not be maintained
		2	Dual feasibility is maintained
		3	The sum of infeasibilities is not allowed to increase
		4	Dual feasibility is maintained and the sum of infeasibilities is not allowed to increase
	Note: The options to maintain dual feasibility do not have much impact due to the way GAMS sets up the model. Normally, only options 0 or 1 need be used.		
`creatbas`	character string		Create a basis file in MPS style. String is the file name. Can only be used if GAMS passes on a basis (i.e. not the first solve, and if BRATIO test is passed). The basis is written just after reading in the problem and after the basis is setup. The option `bastype` determines whether values are written to the basis file. **On UNIX precede the filename by ../ due to a bug in OSL (see Section 7.3).**
`creatmps`	character string		This option can be used to create an MPS file of the GAMS model. This can be useful to test out other solvers or to pass on problems to the developers. String is file name. Its should not be longer than 30 characters. Example: `creatmps trnsport.mps`. The option `mpstype` determines which type (1 or 2 nonzeroes per line) is being used. The MPS file is written just after reading in the problem, before scaling and presolve. **On UNIX precede the filename by ../ due to a bug in OSL (see Section 7.3).** Examples of the MPS files generated by OSL are in the Appendix.

`crossover`	integer		Switching to simplex from the Interior Point method. Use of this option can be expensive. It should be used if there is a need to restart from the optimal solution in the next `solve` statement. *(default = 1)*
		0	Interior point method does not switch to Simplex
		1	Interior point method switches to Simplex when numerical problems arise.
		2	Interior point method switches to Simplex at completion or when in trouble.
		3	As in 2, but also if after analyzing the matrix it seems the model is more appropriate for the Simplex method.
		4	Interior point immediately creates a basis and switches over to Simplex.
`cutoff`	real		Cutoff or incumbent value. Overrides the CUTOFF in GAMS.
`cuts`	integer		Allocate space for extra cuts generated by Branch and Bound preprocessor. *(default = 10 + m/10, where m is the number of rows)*
`degscale`	real		The scale factor for all degradation. Range - [0.0, *maxreal*] *(default = 1.0)*
`densecol`	integer		Dense column threshold. Columns with more non-zeroes than DENSECOL are treated differently. Range - [10, *maxint*] *(default = 99,999)*
`densethr`	real		Density threshold in Cholesky. If DENSETHR ≤ 0 then everything is considered dense, if DENSETHR ≥ 1 then everything is considered sparse. Range - [-maxreal, maxreal] *(default - 0.1)*
`devex`	integer		Devex pricing *(default = 1)*
		0	Switch off Devex pricing (not recommended for normal use).
		1	Approximate Devex method.
		2	Primal: Exact Devex method, Dual: Steepest Edge using inaccurate initial norms.

	3	Exact Steepest Edge method.
	-1,-2,-3	As 1,2,3 for the dual. For the primal the default is used until feasible, and then mode 1,2 or 3.

Note: Devex pricing is only used by the simplex method. Devex pricing is used after the first "cheap" iterations which use a random pricing strategy. In the primal case a negative value tells OSL to use the non-default devex pricing strategy only in phase II (in phase I the default devex strategy is used then). For the primal 1 (the default) or -2 are usually good settings, for the dual 3 is often a good choice.

`Droprowct`	integer	The constraint dropping threshold. Range - [1,30] *(default = 1)*
`dweight`	real	Proportion of the feasible objective that is used when the solution is dual infeasible. Range - [0.0,1.0] *(default = 0.1)*
`factor`	integer	Refactorization frequency. A factorization of the basis matrix will occur at least each `factor` iterations. OSL may decide to factorize earlier based on some heuristics (loss of precision, space considerations) Range - [0,999] *(default = 100 + m/100, where m is the number of rows)*
`fastits`	integer	When positive, OSL switches to Devex after `fastits` random iterations, (to be precise at the first refactorization after that) but before it will price out a random subset, with correct reduced costs. When negative, OSL takes a subset of the columns at each refactorization, and uses this as a working set. Range - [-*maxint*,+*maxint*]. *(default = 0)*

Note: By default OSL primal simplex initially does cheap iterations using a random pricing strategy, then switches to Devex either because the success rate of the random pricing becomes low, or because the storage requirements are becoming high. This option allows you to change this pricing change. This option is only used when simplex was used with `simopt 2` which is the default for models solved from scratch. For more information see the OSL manual.

`fixvar1`	real		Tolerance for fixing variables in the barrier method if the problem is still infeasible. Range - [0.0, 1.0e-3] *(default = 1.0e-7)*
`fixvar2`	real		Tolerance for fixing variables when the problem is feasible. Range - [0.0, 1.0e-3] *(default: 1.0e-8)*
`formntype`	integer		Formationof normal matrix *(default = 0)*
		0	Do formation of normal matrix in the interior point methods in a way that exploits vectorization. Uses lots of memory, but will switch to option 2 if needed.
		1	Save memory in formation of normal matrix.
`Improve`	real		The next integer solution should be at least `improve` better than the current one. Overrides the `cheat` parameter in GAMS.
`Incore`	integer		Keep tree in core *(default = 0)*
		0	The Branch & Bound routine will save tree information on the disk. This is needed for larger and more difficult models.
		1	The tree is kept in core. This is faster, but you may run out of memory. For larger models use `incore 0`, or use the `workspace` model suffix to request lots of memory. GAMS/OSL can not recover if you are running out of memory, i.e. we cannot save the tree to disk, and go on with `incore 0`.
`Iterlim`	integer		Iteration limit. Overrides the GAMS `iterlim` option. Notice that the interior point codes require much less iterations than the simplex method. Most LP models can be solved with the interior point method with 50 iterations. MIP models may often need much more than 1000 LP iterations. *(default = 1000)*

`iterlog`	integer	LP iteration log frequency. How many iterations are performed before a new line to the log file (normally the screen) is written. The log shows the iteration number, the primal infeasibility, and the objective function. The same log frequency is passed on to OSL, so in case you have `option sysout=on` in the GAMS source, you will see an OSL log in the listing file with the same granularity. The resource usage is checked only when an iteration log is written. In case you set this option parameter to a very large value, a resource limit overflow will not be detected. *(default = 20)*
`iweight`	real	Weight for each integer infeasibility. Range [0.0, *maxreal*] *(default = 1.0)*
`maxnodes`	integer	2Maximum number of nodes allowed *(default = 9,999,999)*
`maxprojns`	integer	Maximum number of null space projections. Range - [1,10] *(default = 3)*
`maxsols`	integer	Maximum number of integer solution allowed. *(default = 9,999,999)*
`method`	character	Solution Method *(default = psimplex)*
	psimplex	Uses primal simplex method as LP solver
	dsimplex	Uses dual simplex
	simplex	Uses either the primal or dual simplex based on model characteristics
	network	Uses the network solver
	interior0	Uses an appropriate barrier method
	interior1	Uses the primal barrier method
	interior2	Uses the primal-dual barrier method
	interior3	Uses the primal-dual predictor-corrector barrier method.

Note: In case any of the simplex algorithms is used the listing file will list the algorithm chosen by OSL (this was not possible for `interior0`). the best interior point algorithm overall seems to be the primal-dual with predictor-corrector. Note that if an interior point method has been selected for a MIP problem, only the relaxed LP is solved by the interior point method.

`Mpstype`	integer		Format of MPS file *(default = 2)*
		1	one nonzero per line in the COLUMN section
		2	.two nonzeroes per line in the COLUMN section.
`Mufactor`	real		The reduction factor for μ in the primal barrier method. Range - [1.0e-6,0.99999] *(default = 0.1)*
`muinit`	real		Initial value for μ in primal barrier method. Range - [1.0e-20, 1.0e6] *(default = 0.1)*
`mulimit`	real		Lower limit for μ. Range - [1.0e-6,1.0] *(default = 1.0e-8)*
`mulinfac`	real		Multiple of μ to add to the linear objective. Range - [0.0, 1.0] *(default = 0.0)*
`networklog`	integer		Iteration log frequency for the network solver. See `iterlog`. *(default = 100)*
`nodelog`	integer		MIP node log frequency. Node log is written to the screen when a new integer is found or after `nodelog` nodes. *(default = 20)*

Note: OSL writes a log line each node. This is captured in the status file, and displayed in the listing file when you have `option sysout=on;` in the GAMS source. The status file can become huge when many nodes are being examined.

`Nullcheck`	integer	Null space checking switch. See the OSL manual.
`Objweight`	real	Weight given to true objective function in phase 1 of the primal barrier method. Range - [0.0, 1.0e8] *(default = 0.1)*

`optca`	real		Absolute optimality criterion. The definition of this criterion is the same as described in the GAMS manual. *(default = 0.0)*
`optcr`	real		Relative optimality criterion. The definition of this criterion is the same as described in the GAMS manual. *(default = 0.10)*
`pdgaptol`	real		Barrier method primal-dual gap tolerance. Range - [1.0e-12, 1.0e-1] *(default = 1.0e-7)*
`pdstepmult`	real		Step-length multiplier for pri2mal-dual barrier. Range - [0.01, 0.99999] *(default = 0.99995)*
`pertdiag`	real		Diagonal perturbation in the Cholesky factorization. Range - [0.0, 1.0e-6] *(default = 1.0e-12)*
`possbasis`	integer		Potential basis flag. If greater than zero, variables that are remote from their bounds are marked potentially basic. Range - [0,maxint] *(default = 1)*
`postsolve`	integer		Basis solution required. If the presolver was not used this option is ignored. *(default = 1)*
		0	No basis solution is required. The reported solution will be optimal but it will not be a basis solution. This will require less work than being forced to find a basic optimal solution.
		1	Basis solution is required. After the postsolve, the Simplex method is called again to make sure that the final optimal solution is also a basic solution

`presolve` integer

Perform model reduction before starting optimization procedure. This should not be with network solver. Ifby accident, all the constraints can be removed from the model, OSL will not be able to solve it and will abort. `presolve` can sometimes detect infeasibilities which can cause it to fail, in which case the normal solver algorithm is called for the full problem.

(default = 0 for cold starts, -1 for restarts)

-1	Do not use presolve
0	Remove redundant rows, the variables summing to zero are fixed. If just one variable in a row is not fixed, the row is removed and appropriate bounds are imposed on that variable.
1	As 0, and doubleton rows are eliminated (rows of the form $px_j + qx_k = b$).
2	As 0, and rows of the form $x_1 = x_2 + x_3 ..., x>0$ are eliminated.
3	All of the above are performed.

Note: The listing file will report how successful the presolve was. The presolver writes information needed to restore the original to a diskfile, which is located in the gams scratch directory. The postsolve routine will read this file after the smaller problem is solved, and then simplex is called again to calculate an optimal basis solution of the original model. If no basis solution is required use the option `postsolve 0`.

`Projtol` real

Projection error tolerance.

Range - [0.0, 1.0]
(default 1.0e-6)

`pweight` real

Starting value of the multiplier in composite objective function.
Range: [1e-12,1e10].
(default = 0.1)

Note:OSL uses in phase I when the model is still infeasible a composite objective function of the form *phase_I_objective + pweight * phase_II_objective*. The phase I objective has a +1 or -1 where the basic variables exceed their lower or upperbounds. This gives already a little bit weight to the phase II objective. `pweight` starts with 0.1 (or whatever you specify as starting value) and is decreased continuously. It is decreased by a large amount if the infeasibilities increase and by small amounts if progress to feasibility is slow.

`Reslim`	integer		Resource limit. Overrides the GAMS `reslim` option. *(default = 1000s)*
`rgfactor`	real		Reduced gradient target reduction factor. Range - [1.0e-6,0.999999] *(default = 0.1)*
`rglimit`	real		Reduced gradient limit. Range - [0.0, 1.0] *(default = 0.0)*
`scale`	integer		Scaling done on the model. Scaling cannot be used for network models. It is advised to use scaling when using the primal barrier method, the other barrier methods (primal-dual and primal-dual predictor-corrector) in general should not be used with scaling. *(default = 1 except for methods mentioned above)*
		0	Do not scale the model
		1	Scale the model
`simopt`	integer		Simplex option *(default = 2 if no basis provided by GAMS and crashing off, = 0 otherwise)*
		0	Use an existing basis
		1	Devex pricing is used
		2	Random pricing is used for the first "fast" iterations, then a switch is made to Devex pricing
		3	Use previous values to reconstruct a basis.
	Note: The dual simplex method can only have options 0 or 1.		
`stepmult`	real		Step-length multiplier for primal barrier algorithm. Range - [0.01, 0.99999] *(default = 0.99)*
`strategy`	integer		MIP strategy for deciding branching.

1	Perform probing only on satisfied 0-1 variables. This is the default setting in OSL. When a 0-1 variable is satisfied, OSL will do probing to determine what other variables can be fixed as a result. If this bit is not set, OSL will perform probing on all 0-1 variables. If they are still fractional, OSL will try setting them both ways and use probing to build an implication list for each direction.
2	Use solution strategies that assume a valid integer solution has been found. OSL uses different strategies when looking for the first integer solution than when looking for a better one. If you already have a solution from a previous run and have set a cutoff value, this option will cause OSL to operate as though it already has an integer solution. This is beneficial for restarting and should reduce the time necessary to reach the optimal integer solution.
4	Take the branch opposite the maximum pseudo-cost. Normally OSL will branch on the node whose smaller pseudo-cost is highest. This has the effect of choosing a node where both branches cause significant degradation in the objective function, probably allowing the tree to be pruned earlier. With this option, OSL will branch on the node whose larger pseudo-cost is highest. The branch taken will be in the opposite direction of this cost. This has the effect of forcing the most nearly integer values to integers earlier and may be useful when any integer solution is desired, even if not optimal. Here the tree could potentially grow much larger, but if the search is successful and any integer solution is adequate, then most of it will never have to be explored.
8	Compute new pseudo-costs as variables are branched on. Pseudo-costs are normally left as is during the solution process. Setting this option will cause OSL to make new estimates, using heuristics, as each branch is selected.

	16	Compute new pseudo-costs for unsatisfied variables. Pseudo-costs are normally left as is during the solution process. Setting this option will cause OSL to make new estimates, using heuristics, for any unsatisfied variables' pseudo-costs in both directions. This is done only the first time the variable is found to be unsatisfied. In some cases, variables will be fixed to a bound by this process, leading to better performance in the branch and bound routine. This work is equivalent to making two branches for every variable investigated.
	32	Compute pseudo-costs for satisfied variables as well as unsatisfied variables. Here, if 16 is also on, OSL will compute new estimated pseudo-costs for the unsatisfied as well as the unsatisfied ones. Again, this is very expensive, but can improve performance on some problems.
	Note: `strategy` can be a combination of the above options by adding up the values for the individual options. 48 for instance will select 16 and 32.	
`Target`	real	Target value for the objective. *(default = 5% worse than the relaxed solution)*
`threshold`	real	Supernode processing threshold. Range [0.0, *maxreal*] *(default = 0)*
`tolpinf`	real	Primal infeasibility tolerance. Row and column levels less than this values outside their bounds are still considered feasible. Range: [1e-12,1e-1]. *(default = 1e-8)*
`toldinf`	real	Dual infeasibility tolerance. Functions as optimality tolerance for primal simplex. Range: [1e-12,1e-1]. *(default = 1e-7)*
`tolint`	real	Integer tolerance Range - [1.0e-12, 1.0e-1] *(default: 1.0e-6)*
`workspace`	real	Memory allocation. Overrides the memory estimation and the `workspace` model suffix. Workspace is defined in Megabytes.

7 SPECIAL NOTES

This section covers some special topics of interest1 to users of OSL.

7.1 Cuts generated during branch and bound

The OSL documentation does not give an estimate for the number of cuts that can be generated when the `bbpreproc` is used. By default we now allow #rows/10 extra rows to be added. Use the `cuts` option to change this. Also we were not able to find out how many cuts OSL really adds, so we can not report this. You will see a message on the screen and in the listing file when OSL runs out of space to add cuts. When this happens, I have seen the branch & bound fail later on with the message: *OSL data was overwritten*, so make sure your `cuts` option is large enough. For this reason we have as a default: no preprocessing.

Note that when `cuts` is 0 and `presolve` is turned on, there may still be enough room for the preprocessor to add cuts, because the presolve reduced the number of rows.

7.2 Presolve procedure removing all constraints from the model

The PRESOLVE can occasionally remove all constraints from a model. This is the case for instance for the first solve in [WESTMIP]. An artificial model with the same behavior is shown below. OSL will not recover from this. Turn off the presolve procedure by using the option `presolve -1` to prevent this from happening. An appropriate message is written when this happens.

```
Variable x ;
Equation e ;
e..  x =e= 1 ;
model m /all/ ;
option lp = osl ;
solve m using lp minimizing x ;
```

7.3 Deleting the tree files

The MIP solver EKKMSLV uses two files to store the tree. Due to a bug in OSL we are unable to store these files in the GAMS scratch directory (the open statement we issue is ignored by OSL). Therefore after you solved a MIP with OSL with INCORE 0 (the default), two files *`fort.82`* and *`fort.83`* would be in the current directory. The shell script *`gamsosl.run`* will try to overcome this by doing a **cd** to the scratch directory. If this fails, you will see an error message on the screen.

An undesirable side effect of this is that all options that relate to user specified files have to be preceded by `../` to have the file going to the directory where the user started gams from. If you do not do this, the file will go to the current directory, which is the scratch directory, and the file will be removed when GAMS terminates. The affected commands are `creatmps` and `creatmbas`. So if you want to write an MPS file with the name *`myfile.mps`*, use the option `creatmps ../myfile.mps`.

8 THE GAMS/OSL LOG FILE

The iteration log file that normally appears on the screen has the following appearance on the screen for LP problems:

```
Reading data...
 Starting OSL...
 Scale...
 Presolve...
 Crashing...
 Primal Simplex...
    Iter             Objective   Sum Infeasibilities
      20        2155310.000000          48470.762716
      40        1845110.000000          37910.387877
      60        1553010.000000          26711.895409
      80         191410.000000                   0.0
     100         280780.000000                   0.0
     120         294070.000000                   0.0
 Postsolve...
 Primal Simplex...
     121         294070.000000     Normal Completion
                                   Optimal
```

For MIP problems, similar information is provided for the relaxed problem, and in addition the branch and bound information is provided at regular intervals. The screen log has the following appearance:

```
Reading data...
 Starting OSL...
 Scale...
 Presolve...
 Crashing...
 Primal Simplex...
    Iter             Objective   Sum Infeasibilities
      20             19.000000              8.500000
      40              9.500000              6.250000
 **** Not much progress: perturbing the problem
      60              3.588470              3.802830
      80              0.500000              2.000000
     100              2.662888              0.166163
     116              0.000000     Relaxed Objective
Branch&Bound...
    Iter  Nodes  Rel Gap       Abs Gap    Best Found
     276     19   n/a           6.0000        6.0000 New
     477     39   n/a           6.0000        6.0000
     700     59   n/a           6.0000        6.0000
     901     79   n/a           6.0000        6.0000
```

```
    1119     99  65.0000        5.9091        6.0000
    1309    119  65.0000        5.9091        6.0000
    1538    139  17.0000        5.6667        6.0000
    1701    159  17.0000        5.6667        6.0000
    1866    179  17.0000        5.6667        6.0000
    2034    199  17.0000        5.6667        6.0000
    2158    219  11.0000        5.5000        6.0000
    2362    239  11.0000        5.5000        6.0000
    2530    259   8.0000        5.3333        6.0000
    2661    275   5.0000        3.3333        4.0000 Done
Postsolve...
 Fixing integer variables...
 Primal Simplex...
    2661              4.000000     Normal Completion
                                   Integer Solution
The solution satisfies the termination tolerances
```

The branch and bound information consists of the number of iterations, the number of nodes, the current relative gap, the current absolute gap and the current best integer solution.

9 EXAMPLES OF MPS FILES WRITTEN BY GAMS/OSL.

This appendix shows the different output formats for MPS and basis files. We will not explain the MPS format or the format of the basis file: we will merely illustrate the function of the options `mpstype` and `bastype`. Running [TRNSPORT] with the following option file

```
mpsfile trnsport.mps
```

will result in the following MPS file:

```
NAME          GAMS/OSL
ROWS
 N  OBJECTRW
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
COLUMNS
    C0000001  R0000001     -0.225000   R0000002      1.000000
    C0000001  R0000004      1.000000
    C0000002  R0000001     -0.153000   R0000002      1.000000
    C0000002  R0000005      1.000000
    C0000003  R0000001     -0.162000   R0000002      1.000000
    C0000003  R0000006      1.000000
    C0000004  R0000001     -0.225000   R0000003      1.000000
    C0000004  R0000004      1.000000
    C0000005  R0000001     -0.162000   R0000003      1.000000
    C0000005  R0000005      1.000000
    C0000006  R0000001     -0.126000   R0000003      1.000000
    C0000006  R0000006      1.000000
    C0000007  OBJECTRW      1.000000   R0000001      1.000000
RHS
    RHS1      R0000002    350.000000   R0000003    600.000000
    RHS1      R0000004    325.000000   R0000005    300.000000
    RHS1      R0000006    275.000000
BOUNDS
 FR BOUND1    C0000007      0.000000
ENDATA
```

MPS names have to be 8 characters or less. GAMS names can be much longer, for instance: X("Seattle","New-York"). We don't try to make the names recognizable, but just give them labels like R0000001 etc.

Setting the option `mpstype 1` gives:

```
NAME          GAMS/OSL
ROWS
 N  OBJECTRW
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
COLUMNS
    C0000001  R0000001     -0.225000
    C0000001  R0000002      1.000000
    C0000001  R0000004      1.000000
    C0000002  R0000001     -0.153000
    C0000002  R0000002      1.000000
    C0000002  R0000005      1.000000
    C0000003  R0000001     -0.162000
    C0000003  R0000002      1.000000
    C0000003  R0000006      1.000000
    C0000004  R0000001     -0.225000
    C0000004  R0000003      1.000000
    C0000004  R0000004      1.000000
    C0000005  R0000001     -0.162000
    C0000005  R0000003      1.000000
    C0000005  R0000005      1.000000
    C0000006  R0000001     -0.126000
    C0000006  R0000003      1.000000
    C0000006  R0000006      1.000000
    C0000007  OBJECTRW      1.000000
    C0000007  R0000001      1.000000
RHS
    RHS1      R0000002    350.000000
    RHS1      R0000003    600.000000
    RHS1      R0000004    325.000000
    RHS1      R0000005    300.000000
    RHS1      R0000006    275.000000
BOUNDS
 FR BOUND1    C0000007      0.000000
ENDATA
```

To illustrate the creation of a basis file, we first solve the transport model as usual, but we save work files so we can restart the job:

```
gams trnsport save=t
```

Then we create a new file called *t2.gms* with the following content:

```
transport.optfile=1;
solve trnsport using lp minimizing z;
```

and we run *gams t2 restart=t* after creating an option file containing the line `creatbas trnsport.bas`. This results in the following basis file being generated.

```
NAME
 XL C0000002  R0000001
 XU C0000004  R0000004
 XU C0000006  R0000005
 XU C0000007  R0000006
ENDATA
```

When we change the option to bastype 1 we get:

```
NAME
 BS R0000002               0.000000
 BS R0000003               0.000000
 LL C0000001               0.000000
 XL C0000002  R0000001     0.000000                 0.000000
 LL C0000003               0.000000
 XU C0000004  R0000004     0.000000               325.000000
 LL C0000005               0.000000
 XU C0000006  R0000005     0.000000               300.000000
 XU C0000007  R0000006     0.000000               275.000000
ENDATA
```

GAMS/OSL Stochastic Extensions User Notes

Table of Contents

Introduction

This document describes the GAMS 2.50 link to the IBM OSL Stochastic Extensions solvers.

The GAMS/OSLSE link combines the high level modeling capabilities of GAMS with a powerful decomposition solver for stochastic linear and quadratic programs. It allows the user to formulate the deterministic equivalent stochastic program using general GAMS syntax and to pass this on to the solver without having to be concerned with the programming details of such a task. The default solver options used, while suitable for most solves, can be changed via a simple options file.

Those seeking an introduction to the ideas and techniques of stochastic programming should consult the IBM Stochastic Extensions web page or the recently published texts *Introduction to Stochastic Programming* (J.R. Birge and F.V. Louveaux, Springer-Verlag, New York, 1997) or *Stochastic Programming* (P. Kall and S.W. Wallace, John Wiley and Sons, Chichester, England, 1994).

How to run a model with GAMS/OSLSE

The following statements can be used inside your GAMS model:

```
option lp  = oslse;   for linear programs, or
option nlp = oslse;   for quadratic programs
```

These statements should appear before the solve statement you wish to affect.

Overview of OSLSE

OSL Stochastic Extensions (OSLSE) allows the input of a stochastic program and its solution via a nested decomposition solver that implements the L-shaped method of Van-Slyke & Wets, a.k.a. Benders decomposition. The stochastic program is assumed to have an event tree structure that is crucial to the efficient input and solution of the problem. In addition to linear programs, a convex separable quadratic term is allowed in the objective function (i.e. a diagonal Q matrix). At this time, only discrete probability distributions are accommodated.

Model formulation

To properly understand and use the GAMS/OSLSE link, one must understand the *event tree* representation of a stochastic program. The OSLSE subroutine library assumes that the SP has this form, and depends on this fact to efficiently store and solve the problem.

To construct an event tree, the variables in the SP are first partitioned, where the variables in each subset represent decisions made at a particular time period and based on one realization of the stochastic parameters known at that time period. Each of these subsets form a *node*. Thus, for a two-stage SP with 2 possible realizations of a stochastic parameter, there will be one node at time period 1, and two nodes at time period 2, where the variables in the last two nodes represent the same decision made in the face of different data.

Given the set of nodes and the variables belonging to them, we partition the set of constraints or equations in the problem according to this rule: a constraint belongs to the node of latest time stage containing a variable used in the constraint. Put more simply, the constraint goes in the latest node possible. Note that for valid stochastic programs, this rule is not ambiguous: a constraint cannot involve variables in different nodes of the same time period. Thus, a two-stage SP may have constraints involving only those variables from the node at time period 1 (e.g. initial investmest limitations, etc.) that are also assigned to this node, and constraints involving first- and second-stage variables (e.g. recourse constraints) that are assigned to the nodes in time period 2.

Once all variables and equations are partitioned among the nodes, we can define an *ancestor relationship* among the nodes as follows: node *i* is an ancestor of node *j* if an equation in node *j* uses a variable in node *i*. This ancestor relationship does not explicitly define a tree, but in a valid stochastic program, it does so implicitly. The *root* of the tree is the node (at time period 1) that has no ancestors, while the nodes at time period 2 are descendents of this root node and of no others, so they can be made children of the root.

Intended use

The GAMS/OSLSE link is intended to be used for stochastic models with 2 or more stages, where decisions (variables) from one time stage determine the feasibility or optimality of decisions made in later time stages. The key requirement is that the modeler group the variables and constraints by node. If this is done correctly, the link takes care of all the other details. The quickest and easiest way to begin using the GAMS/OSLSE link is to meet this key requirement and follow the examples below. Those wishing to push the limits of the link or to understand the error messages for bad formulations, etc., will want to read the next subsection on model requirements more carefully.

The modeler must identify one set in the GAMS model as the *node set*. This is done by using the descriptive text `nodes` (not case sensitive) for the set in question. The set name and set elements can be any legal GAMS values. Each equation and variable in the model (with the exception of the objective) can then be indexed by this node set, so that the link can partition the set of rows and columns by node and construct an event tree for the model. Note that model formulation that results is completely general, so that it can be passed to any LP or NLP solver. Only the OSLSE solver is making use of this special partitioning information.

As an example, consider the simple two-stage model (the stochastic nature is suppressed for simplicity):

$$\begin{aligned} \textbf{max} \quad & x_i && i = 1, 2 \\ \text{s.t.} \quad & x_{root} <= 1 \\ & x_i = x_{root} + 1, && i = 1, 2 \end{aligned}$$

An equivalent GAMS model is the following:

```
set
    N 'nodes' / root,one,two /,
    LEAF(N)   / one, two /,
    ROOT(N)   / root /;

variables
    x(N),
    obj;

equations
    objdef,
    g(N),
    f(N);

objdef..     sum {LEAF, x(LEAF)} =e= obj;
f(ROOT)..    x(ROOT) =l= 1;
g(LEAF)..    x(LEAF) =e= x('root') + 1;

model eg1 / f, g, objdef /;
solve eg1 using lp maximizing obj;
```

Note that in the model above, every variable except the objective is indexed by the set N, and similarly for the equations. The objective is treated specially, so that it can involve variables from the same time period and different nodes. It is not used in determining the ancestor relationship between the nodes, and in fact, should never be indexed.

It is not strictly necessary to index all variables and equations by node. Any *nodeless* variables or equations (i.e. those not indexed by node) are placed into the root node of the event tree. So, a completely equivalent formulation for the above model would be:

```
set
    N 'nodes' / root, one,two /,
    LEAF(N)   / one, two /;

variables
    xr          'root variable'
    x(N),
```

```
    obj;

equations
    objdef,
    f,
    g(N);

objdef..      sum {LEAF, x(LEAF)} =e= obj;
g(LEAF)..     x(LEAF) =e= 1 + xr;
f..           xr =l= 1;

model eg1 / f, g, objdef  /;

option lp = oslse;
solve eg1 using lp maximizing obj;
```

There are a number of stochastic programming examples on the OSLSE Web page. Formulations of these and other models in GAMS and suitable for use with GAMS/OSLSE can be downloaded here.

Model requirements

While there is an implicit event tree in all stochastic models of the type we are considering, it is not necessary for the modeler to construct this tree explicitly. In some cases, it is good modeling practice to do so, but this is not a requirement made by GAMS/OSLSE. All that is necessary is that (some) variables and constraints be indexed by node. The link will extract the ancestor relationships between the nodes by looking at the constraints and the variables they use. From these ancestor relationships, a tree is formed, and the resulting model is passed to and solved by the routines in OSLSE.

The above process can fail for a number of reasons. For example, the ancestor relationships between the nodes could be circular (e.g. A -> B -> A), in which case a tree cannot exist. Such a model will be rejected by GAMS/OSLSE. At the opposite extreme, the model could consist of many independent nodes, so that no ancestor relationships exist. In this case, each node is the root of a one-node tree. The link will recognize these multiple roots and create an artificial super-root that adopts each of the root nodes as its children, so that we have a proper tree.

The OSLSE routines make some other assumptions about the correct form of the model that are checked by the link. Each scenario must be complete (i.e. must end in the final time stage), so that all leaf nodes in the tree exist at the same level. Furthermore, each node in a given time stage must have the same nonzero structure. This implies that the number of rows, columns, and nonzeros in these nodes is the

same as well. If any of these counts differ, some diagnostic messages are printed out and the solve is terminated.

There are very few requirements put on the form of the objective function. It must be a scalar (i.e. unindexed) row with an equality relationship in which the objective variable appears linearly. This objective variable must not appear in any other rows of the model. Variables from any node of the model may appear in this row, since the objective row will be removed from the model before the model is passed on for solution.

It is possible to formulate models with separable quadratic terms in the objective. Such quadratic terms must appear only in the objective function. In addition, they must be convex (i.e. positive terms for minimization models, negative for maximization). Furthermore, the quadratic terms in each node of a stage must be identical (i.e. scenarios cannot differ in their quadratic part; all quadratic terms are inherited from the core scenario).

Future Work / Wish List

In this section we list features and functionality that are either incomplete, unfinished, or not yet implemented. They are not listed in any particular order, nor are they necessarily going to be part of any future release. We invite your comments on the importance of these items to you.

Currently, it is not possible to "warm start" the stochastic solvers in OSLSE (i.e. they do not make use of any current information about a basis or level or marginal values).

Currently, it is not possible to return complete basis information to GAMS. The basis status is not made available by the OSLSE link. This is evident on degenerate models, for example: when x is at bound and has a zero reduced cost, is x in the basis or not?

The current link allows the objective function to take a limited quadratic form. The quadratic terms must be separable and must be convex (i.e. coefficients positive for a minimization, negative for a maximization). Furthermore, the quadratic terms on the same variables in different scenarios must be identical. This is in contrast to the linear terms, where the objective weights can change with the scenario. OSLSE only allows quadratic terms to be specified for the core model; new scenarios do not include these terms, but inherit them from the core. The link currently checks for proper agreement between these quadratic terms, and rejects the model if this is not maintained.

The current link does not allow the user to specify an iterations limit or a resource limit, or to cause a

user interrupt. The solution subroutines do not support this. Also, the iteration count is not made available.

When the OSLSE solution routines detect infeasibility or unboundedness, no solution information is passed back (i.e. levels and marginals). This is reported properly to GAMS via the model status indicator.

It is not possible to solve a "trivial" stochastic program, i.e. one with only one stage. The solution subroutines do not allow this. This should be allowed in a future release of OSLSE. Currently, the link rejects single-stage models with an explanatory message.

The number of rows in any stage must be positive. The link checks this and rejects offending models with an explanatory message.

The nonzero structure of each node in a stage must be identical. This is a requirement imposed by OSLSE. It would be possible to construct, in the link, a nonzero structure consisting of the union of all nodes in a stage, but this has not been done in this version. A workaround for this is to use the EPS value in the GAMS model to include zeros into nodes where they did not exist (see the example models for more detail).

GAMS options

There are a number of options one can set in a GAMS model that influence how a solver runs (e.g iteration limit, resource limit). The relevant GAMS options for the OSLSE link are:

`option iterlim = N;`	is used to set the iteration limit. Not clear yet whether this is a limit on Bender's iterations, total pivots, or what.
`option reslim = X;`	is used to set a resource limit (in seconds) for the solver. Currently not implemented.

Summary of GAMS/OSLSE options

The GAMS/OSLSE options are summarized here. The section of detailed descriptions can be found later in this document.

benders_major_iterations	limits # of master problems solved
cut_stage	earliest stage not in master
ekk_crsh_method	OSL crash method
ekk_nwmt_type	OSL matrix compression
ekk_prsl	perform presolve on submodels
ekk_prsl_type	controls reductions attempted in presolve
ekk_scale	perform scaling on submodels
ekk_sslv_algorithm	simplex variant for initial submodel solution
ekk_sslv_init	simplex option for initial submodel solution
large_master_bounds	avoid unboundedness, large swings in master solution
maximum_submodels	limits amount of decomposition
nested_decomp	use nested Benders
optimality_gap	termination criterion
print_tree_all	print nodes, rows, and columns
print_tree_cols	print list of columns in model, grouped by node
print_tree_nodes	print list of nodes in model
print_tree_rows	print list of rows in model, grouped by node
submodel_row_minimum	minimum # of rows in any submodel
write_spl_file	name of SPL file to write

The GAMS/OSLSE log file

This section will not be implemented until the log file format is "finalized".

Detailed description of GAMS/OSLSE options

The options below are valid options for GAMS/OSLSE. To use them, enter the option name and its value, one per line, in a file called `oslse.opt` and indicate you would like to use this options file by including the line <*modname*>`.optfile = 1;` in your GAMS input file or using the `optfile=1` argument on the command line. Is the name of the options file case sensitive? It is only necessary to enter the first three letters of each token of an option name, so the option benders_major_iterations can be entered as ben_maj_iter. Comment lines (those beginning with a #) and blank lines are ignored.

benders_major_iterations *(integer)*

At most this many iterations of Bender's will be done. Another stopping criterion for Bender's is the optimality gap; usually, the gap is reduced to this tolerance before the iteration limit is reached.
(default = 30)

cut_stage *(integer)*

All nodes prior to this time stage are placed in the master problem, while the nodes in this time stage or later are placed into the subproblems. The root node is assumed to be at time stage one. Note that this option keeps nodes in the master, but cannot necessarily keep them out. For example, the master problem will always include one entire scenario, and other scenarios can be added to the master as a result of other options settings, e.g. submodel_row_min or maximum_submodels.
(default = 2)

ekk_crsh_method *(integer)*

Sets type of OSL crash to perform before first solution of submodels.
(default = 3)

0 No crash is performed.
1 Dual feasibility may not be maintained.
2 Dual feasibility is maintained, if possible, by not pivoting in variables that are in the objective function.
3 Dual feasibility may not be maintained, but the sum of the infeasibilities will never increase.
4 Dual feasibility is maintained, if possible, by not pivoting in variables that are in the objective function. In addition, the sum of the infeasibilities will never increase.

ekk_nwmt_type *(integer)*

Controls whether a new, compressed version of the submodel matrices will be created prior to calling the LP submodel solver.
(default = 1)

0 Do not compress.
1 A copy of the matrix is stored by indices.
2 A copy of the matrix is stored by columns.
3 A copy of the matrix is stored in vector block format.
4 A copy of the matrix is stored in vector-block-of-1's format (if the matrix is composed of all 0's and

1's).
5 A copy of the matrix is stored in network format (multiple blocks are compressed into a single network).

ekk_prsl *(yes/no)*

Perform presolve for each submodel of the decomposed problem. Only significant for when using the non-nested decomposition solver (i.e. when nested_decomp is set to no, its default value).
(default = no)

ekk_prsl_type *(integer)*

Controls type of presolve reduction to perform.
(default = 3)

0 Redundant rows are eliminated, positive variables summing to zero are fixed.
1 Type 0 reductions, and the doubleton rows are eliminated.
2 Type 0 reductions, and variable substitutions performed.
3 Type 0, 1, & 2 reductions are done.

ekk_scale *(yes/no)*

Perform scaling for each submodel of the decomposed problem.
(default = no)

ekk_sslv_algorithm *(integer)*

Specifies variant of simplex algorithm to be used for first solution of submodels.
(default = 1)

0 Variant chosen automatically.
1 Primal Simplex.
2 Dual Simplex.

ekk_sslv_init *(integer)*

Simplex option.

(default = 2)

0 Use an existing basis (not recommended).
1 Devex pricing is used.
2 Random pricing is used for the first "fast" iterations, then a switch is made to Devex pricing.
3 Use previous values to reconstruct a basis.

Note:The dual simplex method can only have options 0 or 1.

large_master_bounds *(integer)*

If set to 1, large bounds are placed on the variables in the master problem. This can help to keep the solutions of the master problem from varying too wildly from one iteration to the next.
(default = 0)

maximum_submodels *(integer)*

The problem will be decomposed into at most N submodels. Once this limit is reached, the decomposition code assigns subproblems or nodes to existing submodels. Note that for the (default) nested decomposition solver, the settings of submodel_row_minimum and cut_stage take precedence.
(default = 100)

nested_decomp *(yes/no)*

If yes, use the nested Benders decomposition solution subroutine. Otherwise, use the straight Benders routine.
(default = yes)

optimality_gap *(real)*

The decomposition solvers will termination when the optimality gap is less than this tolerance.
(default = 1e-8)

print_tree_* *(yes/no)*

Prints information about the event tree on the listing file. The node listing indicates what nodes are in the

tree, their parents and children, and what GAMS label is used to index a given node. The row and column listings, printed separately from the node listing, print a list of the rows and columns of the GAMS model, grouped by node.
(default = no)

submodel_row_minimum *(integer)*

If the argument N > 0, nodes are added to a submodel until the number of rows in the submodel exceeds N. This option applies only to the nested decomposition solver. If N > 0, it overrides the setting of cut_stage and maximum_submodels.
(default = 0)

write_spl_file *(string)*

Once the problem has been passed successfully to the OSLSE routines, the problem will be written to a file of the name given as the argument to this option keyword. The filename is not given an SPL extension, but is taken as-is. The SPL file format is a format internal to OSLSE used for compact storage of a particular model.
(SPL file not written by default)

GAMS/PATH User Guide
Version 4.3

Michael C. Ferris
Todd S. Munson

March 20, 2000

Contents

Chapter 1

Complementarity

A fundamental problem of mathematics is to find a solution to a square system of nonlinear equations. Two generalizations of nonlinear equations have been developed, a constrained nonlinear system which incorporates bounds on the variables, and the complementarity problem. This document is primarily concerned with the complementarity problem.

The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. In its simplest form, the combinatorial problem is to choose from $2n$ inequalities a subset of n that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and economics [20] where we might want to compute Wardropian and Walrasian equilibria, and optimization where we can model the first order optimality conditions for nonlinear programs [29, 30]. Other examples, such as bimatrix games [31] and options pricing [27], abound.

Our development of complementarity is done by example. We begin by looking at the optimality conditions for a transportation problem and some extensions leading to the nonlinear complementarity problem. We then discuss a Walrasian equilibrium model and use it to motivate the more general mixed complementarity problem. We conclude this chapter with information on solving the models using the PATH solver and interpreting the results.

1.1 Transportation Problem

The transportation model is a linear program where demand for a single commodity must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs, where $(i,j) \in \mathcal{A}$ means that there is a route from supplier i to demand center j. The problem variables are the quantities $x_{i,j}$ shipped over each arc $(i,j) \in \mathcal{A}$. The linear program can be written mathematically as

$$\begin{array}{rc} \min_{x \geq 0} & \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \\ \text{subject to} & \sum_{j:(i,j)\in\mathcal{A}} x_{i,j} \leq s_i, \ \forall i \\ & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \ \forall j. \end{array} \tag{1.1}$$

where $c_{i,j}$ is the unit shipment cost on the arc (i,j), s_i is the available supply at i, and d_j is the demand at j.

The derivation of the optimality conditions for this linear program begins by associating with each constraint a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. We label the prices on the supply constraint p^s and those on the demand constraint p^d. Intuitively, for each supply node i

$$0 \leq p_i^s, \ s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}.$$

Consider the case when $s_i > \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, that is there is excess supply at i. Then, in a competitive marketplace, no rational person is willing to pay for more supply at node i; it is already over-supplied. Therefore, $p_i^s = 0$. Alternatively, when $s_i = \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, that is node i clears, we might be willing to pay for additional supply of the good. Therefore, $p_i^s \geq 0$. We write these two conditions succinctly as:

$$0 \leq p_i^s \quad \perp \quad s_i \geq \textstyle\sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p_i^s$, the first case, or $s_i = \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, the second case.

Similarly, at each node j, the demand must be satisfied in any feasible solution, that is

$$\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j.$$

Furthermore, the model assumes all prices are nonnegative, $0 \leq p_j^d$. If there is too much of the commodity supplied, $\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} > d_j$, then, in a competitive marketplace, the price p_j^d will be driven down to 0. Summing these relationships gives the following complementarity condition:

$$0 \leq p_j^d \quad \perp \quad \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$

The supply price at i plus the transportation cost $c_{i,j}$ from i to j must exceed the market price at j. That is, $p_i^s + c_{i,j} \geq p_j^d$. Otherwise, in a competitive marketplace, another producer will replicate supplier i increasing the supply of the good in question which drives down the market price. This chain would repeat until the inequality is satisfied. Furthermore, if the cost of delivery strictly exceeds the market price, that is $p_i^s + c_{i,j} > p_j^d$, then nothing is shipped from i to j because doing so would incur a loss and $x_{i,j} = 0$. Therefore,

$$0 \leq x_{i,j} \quad \perp \quad p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i,j) \in \mathcal{A}.$$

We combine the three conditions into a single problem,

$$\begin{array}{rcll} 0 \leq p_i^s & \perp & s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, & \forall i \\ 0 \leq p_j^d & \perp & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, & \forall j \\ 0 \leq x_{i,j} & \perp & p_i^s + c_{i,j} \geq p_j^d, & \forall (i,j) \in \mathcal{A}. \end{array} \tag{1.2}$$

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions [6] of the linear program (1.1). For linear programs the complementary slackness conditions are both necessary and sufficient for x to be an optimal solution of the problem (1.1). Furthermore, the conditions (1.2) are also the necessary and sufficient optimality conditions for a related problem in the variables (p^s, p^d)

$$\begin{array}{ll} \max_{p^s,p^d\geq 0} & \sum_j d_j p_j^d - \sum_i s_i p_i^s \\ \text{subject to} & c_{i,j} \geq p_j^d - p_i^s, \ \forall (i,j) \in \mathcal{A} \end{array}$$

termed the dual linear program (hence the nomenclature "dual variables").

Looking at (1.2) a bit more closely we can gain further insight into complementarity problems. A solution of (1.2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However,

```
sets   i   canning plants,
       j   markets ;

parameter
       s(i)    capacity of plant i in cases,
       d(j)    demand at market j in cases,
       c(i,j)  transport cost in thousands of dollars per case ;

$include transmcp.dat

positive variables
       x(i,j)           shipment quantities in cases
       p_demand(j)      price at market j
       p_supply(i)      price at plant i;

equations
       supply(i)      observe supply limit at plant i
       demand(j)      satisfy demand at market j
       rational(i,j);

supply(i) ..     s(i) =g= sum(j, x(i,j)) ;

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

rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;

model transport / rational.x, demand.p_demand, supply.p_supply /;

solve transport using mcp;
```

Figure 1.1: A simple MCP model in GAMS, `transmcp.gms`

the key to the modeling power of complementarity is that it chooses which of the inequalities in (1.2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active. A regime shift could, for example, be a back stop technology like windmills that become profitable if a CO_2 tax is increased.

1.1.1 GAMS Code

The GAMS code for the complementarity version of the transportation problem is given in Figure 1.1; the actual data for the model is assumed to be given in the file `transmcp.dat`. Note that the model written corresponds very closely to (1.2). In GAMS, the $\perp$ sign is replaced in the `model` statement with a ".". It is precisely at this point that the pairing of variables and equations shown in (1.2) occurs in the GAMS code. For example, the function defined by `rational` is complementary to the variable `x`. To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable $z(i)$):

```
z.lo(i) = 0;
```

or alternatively

```
positive variable z;
```

Further information on the GAMS syntax can be found in [35]. *Note that GAMS requires the modeler to write* `F(z) =g= 0` *whenever the complementary variable is lower bounded, and does not allow the alternative form* `0 =l= F(z)`.

1.1.2 Extension: Model Generalization

While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods [37]), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (1.2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1.1 is independent of the prices p. Since the prices p are variables in the complementarity problem (1.2), we can easily replace the constant demand d with a function $d(p)$ in the complementarity setting. Clearly, any algebraic function of p that can be expressed in GAMS can now be added to the model given in Figure 1.1. For example, a linear demand function could be expressed using

$$\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j(1 - p_j^d), \ \forall j.$$

Note that the demand is rather strange if p_j^d exceeds 1. Other more reasonable examples for $d(p)$ are easily derived from Cobb-Douglas or CES utilities. For

those examples, the resulting complementarity problem becomes nonlinear in the variables p. Details of complementarity for more general transportation models can be found in [13, 16].

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (1.2) is replaced by

$$p_i^s(1+t_i) + c_{i,j} \geq p_j^d, \ \forall (i,j) \in \mathcal{A}.$$

The taxes can be made endogenous to the model, details are found in [35].

The key point is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In many cases, there is no optimization problem corresponding to the complementarity conditions.

1.1.3 Nonlinear Complementarity Problem

We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast as nonlinear complementarity problems (NCPs) defined as follows:

(NCP) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that

$$0 \leq z \perp F(z) \geq 0.$$

Recall that the $\perp$ sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, $z_i F_i(z) = 0$. We frequently refer to this property as z_i is "complementary" to F_i. A special case of the NCP that has received much attention is when F is a linear function, the linear complementarity problem [8].

1.2 Walrasian Equilibrium

A Walrasian equilibrium can also be formulated as a complementarity problem (see [33]). In this case, we want to find a price $p \in \mathbf{R}^m$ and an activity level $y \in \mathbf{R}^n$ such that

$$\begin{array}{rcrcr} 0 \leq y & \perp & L(p) & := & -A^T p \geq 0 \\ 0 \leq p & \perp & S(p,y) & := & b + Ay - d(p) \geq 0 \end{array} \tag{1.3}$$

```
$include walras.dat

positive variables p(i), y(j);
equations S(i), L(j);

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, g(k)*p(k)) / p(i)
         =g= 0;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0;

model walras / S.p, L.y /;
solve walras using mcp;
```

Figure 1.2: Walrasian equilibrium as an NCP, `walras1.gms`

where $S(p, y)$ represents the excess supply function and $L(p)$ represents the loss function. Complementarity allows us to choose the activities y_j to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras' law which states that supply equals demand if we assume all prices p will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter.

1.2.1 GAMS Code

A GAMS implementation of (1.3) is given in Figure 1.2. Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [26].

1.2.2 Extension: Intermediate Variables

In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function $d(p)$ in the Walrasian equilibrium (1.3). The syntax for carrying this out is shown in Figure 1.3 where we use the variables `d` to store the demand function referred to in the excess

```
$include walras.dat

positive variables p(i), y(j);
variables d(i);
equations S(i), L(j), demand(i);

demand(i)..
         d(i) =e= c(i)*sum(k, g(k)*p(k)) / p(i) ;

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;
```

Figure 1.3: Walrasian equilibrium as an MCP, `walras2.gms`

supply equation. The model `walras` now contains a mixture of equations and complementarity constraints. Since constructs of this type are prevalent in many practical models, the GAMS syntax allows such formulations.

Note that positive variables are paired with inequalities, while free variables are paired with equations. A crucial point misunderstood by many experienced modelers is that *the bounds on the variable determine the relationships satisfied by the function F*. Thus in Figure 1.3, d is a free variable and therefore its paired equation `demand` is an equality. Similarly, since p is nonnegative, its paired relationship `S` is a (greater-than) inequality.

A simplification is allowed to the model statement in Figure 1.3. In many cases, it is not significant to match free variables explicitly to equations; we only require that there are the same number of free variables as equations. Thus, in the example of Figure 1.3, the model statement could be replaced by

```
model walras / demand, S.p, L.y /;
```

This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged. GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equations equals the number of remaining free variables. However, if an explicit match

is given, the PATH solver can frequently exploit the information for better solution. Note that all variables that are not free and all inequalities must be explicitly matched.

1.2.3 Mixed Complementarity Problem

A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds ℓ, the upper bounds u and the function F.

(MCP) Given lower bounds $\ell \in \{\mathbf{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbf{R} \cup \{\infty\}\}^n$ and a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that *precisely* one of the following holds for each $i \in \{1, \ldots, n\}$:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

These relationships define a general MCP (sometimes termed a rectangular variational inequality [25]). We will write these conditions compactly as

$$\ell \leq x \leq u \quad \perp \quad F(x).$$

Note that the nonlinear complementarity problem of Section 1.1.3 is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

```
z.lo(I) = 0;
```

or declare

```
positive variable z;
```

Another special case is a square system of nonlinear equations

(NE) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$ find $z \in \mathbf{R}^n$ such that

$$F(z) = 0.$$

In order to formulate this in the GAMS/MCP format we just declare

```
free variable z;
```

In both the above cases, we *must not* modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of F. If a variable z_i is fixed, then $F_i(z)$ is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when $z_i = \ell_i = u_i$. Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time. As an example, in economics, fixing a level of production will remove the zero-profit condition for that activity.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u.$$

The first order optimality conditions for this problem class are precisely MCP($\nabla f(x)$, $[\ell, u]$). We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so $\nabla f(x) \geq 0$. Conversely, if x is at its upper bound, then the function must be increasing as x decreases, so that $\nabla f(x) \leq 0$. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in Figure 1.3 it may be necessary to have an upper bound on the activity level y. In this case, we simply add an upper bound to y in the model statement, and replace the loss equation with the following definition:

```
y.up(j) = 10;
L(j)..   -sum(i, p(i)*A(i,j)) =e= 0 ;
```

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less than inequality or greater than inequality, since this determination depends on the values of the solution variables. We adopt the convention that all bounded variables are paired to equations. Further details on this point are given in Section 1.3.1. However, let us interpret the relationships that the above change generates. If $y_j = 0$, the

loss function can be positive since we are not producing in the jth sector. If y_j is strictly between its bounds, then the loss function must be zero by complementarity; this is the competitive assumption. However, if y_j is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [10, 19, 20].

1.3 Solution

We will assume that a file named `transmcp.gms` has been created using the GAMS syntax which defines an MCP model `transport` as developed in Section 1.1. The modeler has a choice of the complementarity solver to use. We are going to further assume that the modeler wants to use PATH.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the `transmcp.gms` file prior to the `solve` statement

   ```
   option mcp = path;
   ```

   PATH will then be used instead of the default solver provided.

2. Rerun the `gamsinst` program from the GAMS system directory and choose PATH as the default solver for MCP.

To solve the problem, the modeler executes the command:

```
gams transmcp
```

where `transmcp` can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to [4] for further details.

At this stage, control is handed over to the solver which creates a log providing information on what the solver is doing as time elapses. See Chapter 2 for details about the log file. After the solver terminates, a listing file is generated containing the solution to the problem. We now describe the output in the listing file specifically related to complementarity problems.

Code	String	Meaning
1	Normal completion	Solver returned to GAMS without an error
2	Iteration interrupt	Solver used too many iterations
3	Resource interrupt	Solver took too much time
4	Terminated by solver	Solver encountered difficulty and was unable to continue
8	User interrupt	The user interrupted the solution process

Table 1.1: Solver Status Codes

Code	String	Meaning
1	Optimal	Solver found a solution of the problem
6	Intermediate infeasible	Solver failed to solve the problem

Table 1.2: Model Status Codes

1.3.1 Listing File

The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output, an example of which is given in Figure 1.4. We use "..." to indicate where we have omitted continuing similar output.

After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. Table 1.1 and Table 1.2 display the relevant codes that are returned under different circumstances. A modeler can access these codes within the `transmcp.gms` file using `transport.solstat` and `transport.modelstat` respectively.

After this, a listing of the time and iterations used is given, along with a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "****". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal. GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates

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

     MODEL   TRANSPORT
     TYPE    MCP
     SOLVER  PATH                FROM LINE  45

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

 RESOURCE USAGE, LIMIT          0.057     1000.000
 ITERATION COUNT, LIMIT        31        10000
 EVALUATION ERRORS              0            0

 Work space allocated           --    0.06 Mb


---- EQU RATIONAL

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york    -0.225    -0.225     +INF     50.000
seattle  .chicago     -0.153    -0.153     +INF    300.000
seattle  .topeka      -0.162    -0.126     +INF       .

...

---- VAR X           shipment quantities in cases

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york      .       50.000     +INF       .
seattle  .chicago       .      300.000     +INF       .

...

**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
                             0  REDEFINED
                             0     ERRORS
```

Figure 1.4: Listing File for solving `transmcp.gms`

the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value of the equation (an evaluation of the left hand side of the constraint at the current point) should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle  .chicago     -0.153    -2.000     +INF    300.000 INFES
```

The marginal column in the equation contains the value of the the variable that was matched with this equation.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of equ.l - equ.lower and equ.l - equ.upper, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. Figure 1.4 is a good case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

1.3.2 Redefined Equations

Unfortunately, this is not the end of the story. Some equations may have the following form:

```
                LOWER     LEVEL      UPPER    MARGINAL

new-york    325.000   350.000    325.000   0.225 REDEF
```

This should be construed as a warning from GAMS, as opposed to an error. In principle, the `REDEF` should only occur if the paired variable to the constraint had a finite lower and upper bound and the variable is at one of those bounds. In this case, at the solution of the complementarity problem the "equation (=e=)" may not be satisfied. The problem occurs because of a limitation in the GAMS syntax for complementarity problems. The GAMS equations are used to define the function F. The bounds on the function F

```
variables x;
equations d_f;

x.lo = 0;
x.up = 2;

d_f.. 2*(x - 1) =e= 0;

model first / d_f.x /;
solve first using mcp;
```

Figure 1.5: First order conditions as an MCP, `first.gms`

are derived from the bounds on the associated variable. Before solving the problem, for finite bounded variables, we do not know if the associated function will be positive, negative or zero at the solution. Thus, we do not know whether to define the equation as "=e=", "=l=" or "=g=". GAMS therefore allows any of these, and informs the modeler via the "REDEF" label that internally GAMS has redefined the bounds so that the solver processes the correct problem, but that the solution given by the solver does not satisfy the original bounds. However, in practice, a `REDEF` can also occur when the equation is defined using "=e=" and the variable has a single finite bound. This is allowed by GAMS, and as above, at a solution of the complementarity problem, the variable is at its bound and the function F does not satisfy the "=e=" relationship.

Note that this is not an error, just a warning. The solver has solved the complementarity problem specified by this equation. GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the bounds, not from the equation definition.

1.4 Pitfalls

As indicated above, the ordering of an equation is important in the specification of an MCP. Since the data of the MCP is the function F and the bounds ℓ and u, it is important for the modeler to pass the solver the function F and not $-F$.

For example, if we have the optimization problem,

$$\min_{x \in [0,2]} (x-1)^2$$

then the first order optimality conditions are

$$0 \leq x \leq 2 \quad \perp \quad 2(x-1)$$

which has a unique solution, $x = 1$. Figure 1.5 provides correct GAMS code for this problem. However, if we accidentally write the valid equation

```
d_f.. 0 =e= 2*(x - 1);
```

the problem given to the solver is

$$0 \leq x \leq 2 \quad \perp \quad -2(x-1)$$

which has three solutions, $x = 0$, $x = 1$, and $x = 2$. This problem is in fact the stationary conditions for the nonconvex quadratic problem,

$$\max_{x \in [0,2]} (x-1)^2,$$

not the problem we intended to solve.

Continuing with the example, when x is a free variable, we might conclude that the ordering of the equation is irrelevant because we always have the equation, $2(x-1) = 0$, which does not change under multiplication by -1. In most cases, the ordering of equations (which are complementary to free variables) does not make a difference since the equation is internally "substituted out" of the model. In particular, for defining equations, such as that presented in Figure 1.3, the choice appears to be arbitrary.

However, in difficult (singular or near singular) cases, the substitution cannot be performed, and instead a perturbation is applied to F, in the hope of "(strongly) convexifying" the problem. If the perturbation happens to be in the wrong direction because F was specified incorrectly, the perturbation actually makes the problem less convex, and hence less likely to solve. Note that determining which of the above orderings of the equations makes most sense is typically tricky. One rule of thumb is to check whether if you replace the "=e=" by "=g=", and then increase "x", is the inequality intuitively more likely to be satisfied. If so, you probably have it the right way round, if not, reorder.

Furthermore, underlying model convexity is important. For example, if we have the linear program

$$\begin{array}{rl} \min_x & c^T x \\ \text{subject to} & Ax = b,\ x \geq 0 \end{array}$$

we can write the first order optimality conditions as either

$$\begin{array}{rcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & Ax - b \end{array}$$

or, equivalently,

$$\begin{array}{rcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & b - Ax \end{array}$$

because we have an equation. The former is a linear complementarity problem with a positive semidefinite matrix, while the latter is almost certainly indefinite. Also, if we need to perturb the problem because of numerical problems, the former system will become positive definite, while the later becomes highly nonconvex and unlikely to solve.

Finally, users are strongly encouraged to match equations and free variables when the matching makes sense for their application. Structure and convexity can be destroyed if it is left to the solver to perform the matching. For example, in the above example, we could loose the positive semidefinite matrix with an arbitrary matching of the free variables.

Chapter 2

PATH

Newton's method, perhaps the most famous solution technique, has been extensively used in practice to solve to square systems of nonlinear equations. The basic idea is to construct a local approximation of the nonlinear equations around a given point, x^k, solve the approximation to find the Newton point, x^N, update the iterate, $x^{k+1} = x^N$, and repeat until we find a solution to the nonlinear system. This method works extremely well close to a solution, but can fail to make progress when started far from a solution. To guarantee progress is made, a line search between x^k and x^N is used to enforce sufficient decrease on an appropriately defined merit function. Typically, $\frac{1}{2}\left\|F(x)\right\|^2$ is used.

PATH uses a generalization of this method on a nonsmooth reformulation of the complementarity problem. To construct the Newton direction, we use the normal map [34] representation

$$F(\pi(x)) + x - \pi(x)$$

associated with the MCP, where $\pi(x)$ represents the projection of x onto $[\ell, u]$ in the Euclidean norm. We note that if x is a zero of the normal map, then $\pi(x)$ solves the MCP. At each iteration, a linearization of the normal map, a linear complementarity problem, is solved using a pivotal code related to Lemke's method.

Versions of PATH prior to 4.x are based entirely on this formulation using the residual of the normal map

$$\left\|F(\pi(x)) + x - \pi(x)\right\|$$

as a merit function. However, the residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. PATH 4.x considers an alternative nonsmooth system [21], $\Phi(x) = 0$, where $\Phi_i(x) = \phi(x_i, F_i(x))$ and $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$. The merit function, $\|\Phi(x)\|^2$, in this case is differentiable, and is used for globalization purposes. When the subproblem solver fails, a projected gradient direction for this merit function is searched. It is shown in [14] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP is provided under appropriate conditions.

The remainder of this chapter details the interpretation of output from PATH and ways to modify the behavior of the code. To this end, we will assume that the modeler has created a file named `transmcp.gms` which defines an MCP model `transport` as described in Section 1.1 and is using PATH 4.x to solve it. See Section 1.3 for information on changing the solver.

2.1 Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically produced. An example of the log for a particular run is given in Figure 2.1 and Figure 2.2.

The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If PATH is directed to read an option file (see Section 2.4 below), then the following output is generated after the PATH banner.

```
Reading options file PATH.OPT
 > output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.
```

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts working on the problem.

```
--- Starting compilation
--- trnsmcp.gms(46) 1 Mb
--- Starting execution
--- trnsmcp.gms(27) 1 Mb
--- Generating model transport
--- trnsmcp.gms(45) 1 Mb
---    11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
 Work space allocated           --    0.06 Mb
 Reading the matrix.
 Reading the dictionary.
Path v4.3: GAMS Link ver037, SPARC/SOLARIS
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.3 (Sat Feb 26 09:38:08 2000)
Written by Todd Munson, Steven Dirkse, and Michael Ferris

INITIAL POINT STATISTICS
Maximum of X. . . . . -0.0000e+00 var: (x.seattle.new-york)
Maximum of F. . . . . 6.0000e+02 eqn: (supply.san-diego)
Maximum of Grad F . . . . . 1.0000e+00 eqn: (demand.new-york)
                                       var: (x.seattle.new-york)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . 3.0000e+00 eqn: (supply.seattle)
Minimum Row Norm. . . . . 2.0000e+00 eqn: (rational.seattle.new-york)
Maximum Column Norm . . . . . 3.0000e+00 var: (p_supply.seattle)
Minimum Column Norm . . . . . 2.0000e+00 var: (x.seattle.new-york)

Crash Log
major  func  diff  size  residual    step       prox   (label)
    0     0             1.0416e+03             0.0e+00 (demand.new-york)
    1     1     3     3 1.0029e+03  1.0e+00    1.0e+01 (demand.new-york)
pn_search terminated: no basis change.
```

Figure 2.1: Log File from PATH for solving `transmcp.gms`

```
Major Iteration Log
major minor  func  grad  residual    step  type prox    inorm  (label)
    0     0     2     2 1.0029e+03           I 9.0e+00 6.2e+02 (demand.new-york)
    1     1     3     3 8.3096e+02  1.0e+00 SO 3.6e+00 4.5e+02 (demand.new-york)

...

   15     2    17    17 1.3972e-09  1.0e+00 SO 4.8e-06 1.3e-09 (demand.chicago)

FINAL STATISTICS
Inf-Norm of Complementarity . .  1.4607e-08 eqn: (rational.seattle.chicago)
Inf-Norm of Normal Map. . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Minimum Map . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Fischer Function. .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Grad Fischer Fcn. .  1.3247e-09 eqn: (rational.seattle.chicago)

FINAL POINT STATISTICS
Maximum of X. . . . . .  3.0000e+02 var: (x.seattle.chicago)
Maximum of F. . . . . .  5.0000e+01 eqn: (supply.san-diego)
Maximum of Grad F . . . . . .  1.0000e+00 eqn: (demand.new-york)
                                          var: (x.seattle.new-york)

 ** EXIT - solution found.

Major Iterations. . . . 15
Minor Iterations. . . . 31
Restarts. . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . 0.020000
Residual. . . . . 1.397183e-09
--- Restarting execution
```

Figure 2.2: Log File from PATH for solving `transmcp.gms` (continued)

2.1.1 Diagnostic Information

Some diagnostic information is initially generated by the solver at the starting point. Included is information about the initial point and function evaluation. The log file here tells the value of the largest element of the starting point and the variable where it occurs. Similarly, the maximum function value is displays along with the equation producing it. The maximum element in the gradient is also presented with the equation and variable where it is located.

The second block provides more information about the Jacobian at the starting point. These can be used to help scale the model. See Chapter 3 for complete details.

2.1.2 Crash Log

The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in [12], and an exaple of the Crash Log can be seen in Figure 2.1. The first column of the crash log is just a label indicating the current iteration number, the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See Section 3.2.1 for further information. The column "step" corresponds to the steplength taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section 2.4 gives mechanisms to affect the behavior of these steps.

2.1.3 Major Iteration Log

After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see Figure 2.2). The columns that have the same

Code	Meaning
C	A cycle was detected.
E	An error occurred in the linear solve.
I	The minor iteration limit was reached.
N	The basis became singular.
R	An unbounded ray was encountered.
S	The linear subproblem was solved.
T	Failed to remain within tolerance after factorization was performed.

Table 2.1: Linear Solver Codes

labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [31]. The number of pivots performed `per` major iteration is given in the "minor" column.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be taken, due to the use of nonmonotone searches [11, 15], which are used to improve robustness. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. Table 2.1 explains the return codes for the linear solver and Table 2.2 explains the meaning of each step type. The ideal output in this column is either "SO", with "SD" and "SB" also being reasonable. Codes different from these are not catastrophic, but typically indicate the solver is having difficulties due to numerical issues or nonconvexities in the model.

2.1.4 Minor Iteration Log

If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. A listing from `transmcp` model follows, where we have set the `output_minor_iteration_frequency` option to 1.

```
Minor Iteration Log
minor     t          z     w     v   art ckpts  enter    leave
```

Code	Meaning
B	A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.
D	The step was accepted because the Distance between the current iterate and the Newton point was small.
G	A gradient step was performed.
I	Initial information concerning the problem is displayed.
M	The step was accepted because the Merit function value is smaller than the nonmonotone reference value.
O	A step that satisfies both the distance and merit function tests.
R	A Restart was carried out.
W	A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the nonmonotone reference value (M, O, or B step), regenerated the Newton point, and performed a backtracking search.

Table 2.2: Step Type Codes

```
1  4.2538e-01     8     2     0     0     0 t[    0] z[   11]
2  9.0823e-01     8     2     0     0     0 w[   11] w[   10]
3  1.0000e+00     9     2     0     0     0 z[   10] t[    0]
```

`t` is a parameter that goes from zero to 1 for normal starts in the pivotal code. When the parameter reaches 1, we are at a solution to the subproblem. The `t` column gives the current value for this parameter. The next columns report the current number of problem variables z and slacks corresponding to variables at lower bound w and at upper bound v. Artificial variables are also noted in the minor log, see [17] for further details. Checkpoints are times where the basis matrix is refactorized. The number of checkpoints is indicated in the `ckpts` column. Finally, the minor iteration log displays the entering and leaving variables during the pivot sequence.

2.1.5 Restart Log

The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When it is determined that no progress is being made, the

problem is restarted from the initial point supplied in the GAMS file with a different set of options. These restarts give the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

The exact options set during the restart are given in the restart log, part of which is reproduced below.

```
Restart Log
proximal_perturbation 0
crash_method none
crash_perturb yes
nms_initial_reference_factor 2
proximal_perturbation 1.0000e-01
```

If a particular problem solves under a restart, a modeler can circumvent the wasted computation by setting the appropriate options as shown in the log. Note that sometimes an option is repeated in this log. In this case, it is the last option that is used.

2.1.6 Solution Log

A solution report is now given by the algorithm for the point returned. The first component is an evaluation of several different merit functions. Next, a display of some statistics concerning the final point is given. This report can be used detect problems with the model and solution as detailed in Chapter 3.

At the end of the log file, summary information regarding the algorithm's performance is given. The string "** EXIT - solution found." is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what `modelstat` and `solstat` will be returned to GAMS. After this, the "Restarting execution" flag indicates that GAMS has been restarted and is processing the results passed back by PATH.

2.2 Status File

If for some reason the PATH solver exits without writing a solution, or the `sysout` flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but

provides more detailed information. The modeler is typically not interested in this output.

2.3 User Interrupts

A user interrupt can be effected by typing Ctrl-C. We only check for interrupts every major iteration. If a more immediate response is wanted, repeatedly typing Ctrl-C will eventually kill the job. The number needed is controlled by the `interrupt_limit` option. In this latter case, when a kill is performed, no solution is written and an execution error will be generated in GAMS.

2.4 Options

The default options of PATH should be sufficient for most models; the technique for changing these options are now described. To change the default options on the model `transport`, the modeler is required to write a file `path.opt` in the working directory and either add a line

```
transport.optfile = 1;
```

before the `solve` statement in the file `transmcp.gms`, or use the command-line option

```
gams transmcp optfile=1
```

Unless the modeler has changed the WORKDIR parameter explicitly, the working directory will be the directory containing the model file.

We give a list of the available options along with their defaults and meaning in Table 2.3, Table 2.4, and Table 2.5. Note that only the first three characters of every word are significant.

GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the solve statement

```
option iterlim = 2000;
transport.iterlim = 2000;
```

Option	Default	Explanation
convergence_tolerance	1e-6	Stopping criterion
crash_iteration_limit	50	Maximum iterations allowed in crash
crash_merit_function	fischer	Merit function used in crash method
crash_method	pnewton	pnewton or none
crash_minimum_dimension	1	Minimum problem dimension to perform crash
crash_nbchange_limit	1	Number of changes to the basis allowed
crash_perturb	yes	Perturb the problem using pnewton crash
crash_searchtype	line	Searchtype to use in the crash method
cumulative_iteration_limit	10000	Maximum minor iterations allowed
gradient_searchtype	arc	Searchtype to use when a gradient step is taken
gradient_step_limit	5	Maximum number of gradient step allowed before restarting
interrupt_limit	5	Ctrl-C's required before killing job
lemke_start	automatic	Frequency of lemke starts (automatic, first, always)
major_iteration_limit	500	Maximum major iterations allowed
merit_function	fischer	Merit function to use (normal or fischer)
minor_iteration_limit	1000	Maximum minor iterations allowed in each major iteration
nms	yes	Allow line searching, watchdoging, and nonmonotone descent
nms_initial_reference_factor	20	Controls size of initial reference value
nms_maximum_watchdogs	5	Maximum number of watchdog steps allowed
nms_memory_size	10	Number of reference values kept
nms_mstep_frequency	10	Frequency at which m steps are performed

Table 2.3: PATH Options

Option	Default	Explanation
nms_searchtype	line	Search type to use (line, or arc)
output	yes	Turn output on or off. If output is turned off, selected parts can be turned back on.
output_crash_iterations	yes	Output information on crash iterations
output_crash_iterations_frequency	1	Frequency at which crash iteration log is printed
output_errors	yes	Output error messages
output_factorization_singularities	yes	Output linearly dependent columns determined by factorization
output_final_degeneracy_statistics	no	Print information regarding degeneracy at the solution
output_final_point	no	Output final point returned from PATH
output_final_point_statistics	yes	Output information about the point, function, and Jacobian at the final point
output_final_scaling_statistics	no	Display matrix norms on the Jacobian at the final point
output_final_statistics	yes	Output evaluation of available merit functions at the final point
output_final_summary	yes	Output summary information
output_initial_point	no	Output initial point given to PATH
output_initial_point_statistics	yes	Output information about the point, function, and Jacobian at the initial point
output_initial_scaling_statistics	yes	Display matrix norms on the Jacobian at the initial point
output_initial_statistics	no	Output evaluation of available merit functions at the initial point

Table 2.4: PATH Options (cont)

Option	Default	Explanation
output_linear_model	no	Output linear model each major iteration
output_major_iterations	yes	Output information on major iterations
output_major_iterations_frequency	1	Frequency at which major iteration log is printed
output_minor_iterations	yes	Output information on minor iterations
output_minor_iterations_frequency	500	Frequency at which minor iteration log is printed
output_model_statistics	yes	Turns on or off printing of all the statistics generated about the model
output_options	no	Output all options and their values
output_preprocess	yes	Output preprocessing information
output_restart_log	yes	Output options during restarts
output_warnings	no	Output warning messages
preprocess	yes	Attempt to preprocess the model
proximal_perturbation	0	Initial perturbation
restart_limit	3	Maximum number of restarts (0 - 3)
return_best_point	yes	Return the best point encountered or the absolute last iterate
time_limit	3600	Maximum number of seconds algorithm is allowed to run

Table 2.5: PATH Options (cont)

Similarly if the solver runs out of memory, then the workspace allocated can be changed using

```
transport.workspace = 20;
```

The above example would allocate 20MB of workspace for solving the model.

Problems with a singular basis matrix can be overcome by using the `proximal_perturbation` option [3], and linearly dependent columns can be output with the `output_factorization_singularities` option. For more information on singularities, we refer the reader to Chapter 3.

As a special case, PATH can emulate Lemke's method [7, 31] for LCP with the following options:

```
crash_method none;
crash_perturb no;
major_iteration_limit 1;
lemke_start first;
nms no;
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [28, 33, 32] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

```
crash_method none;
crash_perturb no;
lemke_start always;
nms_initial_reference_factor 1;
nms_memory size 1;
nms_mstep_frequency 1;
nms_searchtype line;
merit_function normal;
```

Note that `nms_memory_size 1` and `nms_initial_reference_factor 1` turn off the nonmonotone linesearch, while `nms_mstep_frequency 1` turns off watchdoging [5]. `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, while `merit_function normal` tell PATH to use the normal map for calculating the residual.

2.5 PATHC

PATHC uses a different link to the GAMS system with the remaining code identical. PATHC *does not support MPSGE models*, but enables the use

of preprocessing and can be used to solve constrained systems of nonlinear equations. The output for PATHC is identical to the main distribution described in Section 2.1 with additional output for preprocessing. The options are the same between the two versions.

2.5.1 Preprocessing

The preprocessor is work in progress. The exact output in the final version may differ from that given below.

The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. A comprehensive preprocessor has been incorporated into PATHC as developed in [18].

The preprocessor reports its finding with some additional output to the log file. This output occurs before the initial point statistics. An example of the preprocessing on the `forcebsm` model is presented below.

```
Zero:     0 Single:   112 Double:     0 Forced:     0
Preprocessed size: 72
```

The preprocessor looks for special polyhedral structure and eliminates variables using this structure. These are indicated with the above line of text. Other special structure is also detected and reported.

On exit from the algorithm, we must generate a solution for the original problem. This is done during the postsolve. Following the postsolve, the residual using the original model is reported.

```
Postsolved residual: 1.0518e-10
```

This number should be approximately the same as the final residual reported on the presolved model.

2.5.2 Constrained Nonlinear Systems

Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

$$F(z) = 0,\ \ell \leq z \leq u,$$

where typically, the bounds on z are inactive at the solution. This is *not* an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function F. If a finite bound is active at a solution, the corresponding component of F is only constrained to be nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do not satisfy $F(z) = 0$. Only if z^* is a solution of MCP with $\ell < z^* < u$ is it guaranteed that $F(z^*) = 0$.

Internally, PATHC reformulates a constrained nonlinear system of equations to an equivalent complementarity problem. The reformulation adds variables, y, with the resulting problem written as:

$$\begin{array}{rcl} \ell \leq x \leq u & \perp & -y \\ y \text{ free} & \perp & F(x). \end{array}$$

This is the MCP model passed on to the PATH solver.

Chapter 3

Advanced Topics

This chapter discusses some of the difficulties encountered when dealing with complementarity problems. We start off with a very formal definition of a complementarity problem which is used in later sections on merit functions and ill-defined, poorly-scaled, and singular models.

3.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, $F : D \to \mathbf{R}^n$ where $D \subseteq \mathbf{R}^n$ is the domain of F, and possibly infinite lower and upper bounds, ℓ and u. Let $C := \{x \in \mathbf{R}^n \mid \ell \leq x \leq u\}$, a Cartesian product of closed (possibly infinite) intervals. The problem is given as

$$MCP : \text{find } x \in C \cap D \text{ s.t. } \langle F(x),\ y - x\rangle \geq 0,\ \forall y \in C.$$

This formulation is a special case of the variational inequality problem defined by F and a (nonempty, closed, convex) set C. Special choices of ℓ and u lead to the familiar cases of a system of nonlinear equations

$$F(x) = 0$$

(generated by $\ell \equiv -\infty$, $u \equiv +\infty$) and the nonlinear complementarity problem

$$0 \leq x \ \perp \ F(x) \geq 0$$

(generated using $\ell \equiv 0$, $u \equiv +\infty$).

3.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

3.2.1 Merit Functions

A solver for complementarity problems typically employs a merit function to indicate the closeness of the current iterate to the solution set. The merit function is zero at a solution to the original problem and strictly positive otherwise. Numerically, an algorithm terminates when the merit function is approximately equal to zero, thus possibly introducing spurious "solutions".

The modeler needs to be able to determine with some reasonable degree of accuracy whether the algorithm terminated at solution or if it simply obtained a point satisfying the desired tolerances that is not close to the solution set. For complementarity problems, we can provide several indicators with different characteristics to help make such a determination. If one of the indicators is not close to zero, then there is some evidence that the algorithm has not found a solution. We note that if all of the indicators are close to zero, we are reasonably sure we have found a solution. However, the modeler has the final responsibility to evaluate the "solution" and check that it makes sense for their application.

For the NCP, a standard merit function is

$$\|(-x)_+, (-F(x))_+, [(x_i)_+(F_i(x))_+]_i\|$$

with the first two terms measuring the infeasibility of the current point and the last term indicating the complementarity error. In this expression, we use $(\cdot)_+$ to represent the Euclidean projection of x onto the nonnegative orthant, that is $(x)_+ = \max(x, 0)$. For the more general MCP, we can define a similar function:

$$\left\| x - \pi(x), \left[\left(\frac{x_i - \ell_i}{\|\ell_i\| + 1} \right)_+ (F_i(x))_+ \right]_i, \left[\left(\frac{u_i - x_i}{\|u_i\| + 1} \right)_+ (-F_i(x))_+ \right]_i \right\|$$

where $\pi(x)$ represents the Euclidean projection of x onto C. We can see that if we have an NCP, the function is exactly the one previously given and for nonlinear systems of equations, this becomes $\|F(x)\|$.

There are several reformulations of the MCP as systems of nonlinear (non-smooth) equations for which the corresponding residual is a natural merit function. Some of these are as follows:

- Generalized Minimum Map: $x - \pi(x - F(x))$
- Normal Map: $F(\pi(y)) + y - \pi(y)$
- Fischer Function: $\Phi(x)$, where $\Phi_i(x) := \phi(x_i, F_i(x))$ with

$$\phi(a, b) := \sqrt{a + b} - a - b.$$

  Note that $\phi(a, b) = 0$ if and only if $0 \leq a \perp b \geq 0$. A straightforward extension of Φ to the MCP format is given for example in [14].

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map $\min(x, F(x))$. Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [36]. Figure 3.3 in the subsequent section plots all of these merit functions for the ill-defined example discussed therein and highlights the differences between them.

The squared norm of Φ, namely $\Psi(x) := \frac{1}{2} \sum \phi(x_i, F_i)^2$, is continuously differentiable on $\mathbf{R}^n$ provided F itself is. Therefore, the first order optimality conditions for the unconstrained minimization of $\Psi(x)$, namely $\nabla\Psi(x) = 0$ give another indication as to whether the point under consideration is a solution of MCP.

The merit functions and the information PATH provides at the solution can be useful for diagnostic purposes. By default, PATH 4.x returns the best point with respect to the merit function because this iterate likely provides better information to the modeler. As detailed in Section 2.4, the default merit function in PATH 4.x is the Fischer function. To change this behavior the `merit_function` option can be used.

3.2.2 Crashing Method

The crashing technique [12] is used to quickly identify an active set from the user-supplied starting point. At this time, a proximal perturbation scheme [1, 2] is used to overcome problems with a singular basis matrix. The proximal perturbation is introduced in the crash method, when the matrix factored

is determined to be singular. The value of the perturbation is based on the current merit function value.

Even if the crash method is turned off, for example via the option `crash_method none`, perturbation can be added. This is determined by factoring the matrix that crash would have initially formed. This behavior is extremely useful for introducing a perturbation for singular models. It can be turned off by issuing the option `crash_perturb no`.

3.2.3 Nonmontone Searches

The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [23, 24, 15]. In this case we define a reference value, R^k and we use this value in test for sufficient decrease: test:

$$\Psi(x^k + t_k d^k) \leq \mathbf{R}^k + t_k \nabla \Psi(x^k)^T d^k.$$

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

We now need to detail our choice of the reference value. We begin by letting $\{M_1, \ldots, M_m\}$ be a finite set of values initialized to $\kappa\Psi(x^0)$, where κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code and can be modified with the `nms_initial_reference_factor` option; $\kappa = 1$ indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of $\{M_1, \ldots, M_m\}$ (where the code by default uses $m = 10$), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that d^k is the Newton direction, we define $i_0 = \text{argmax } M_i$ and $R^k = M_{i_0}$. After the nonmonotone linesearch rule above finds t_k, we update the memory so that $M_{i_0} = \Psi(x^k + t_k d^k)$, i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let $x^k = x^{\text{best}}$ where x^{best} is the point with the absolute best merit function value encountered so far. We then recalculate $d^k = -\nabla\Psi(x^k)$ using the best point and let $R^k = \Psi(x^k)$. That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set

$M_i = \Psi(x^k + t_k d^k)$ for all $i \in [1, \ldots, m]$. This prevents future iterates from returning to the same problem area.

A watchdog strategy [5] is also available for use in the code. The method employed allows steps to be accepted when they are "close" to the current iterate. Nonmonotonic decrease is enforced every m iterations, where m is set by the `nms_mstep_frequency` option.

3.2.4 Linear Complementarity Problems

PATH solves a linear complementarity problem each major iteration. Let $M \in \Re^{n \times n}$, $q \in \Re^n$, and $B = [l, u]$ be given. $(\bar{z}, \bar{w}, \bar{v})$ solves the linear mixed complementarity problem defined by M, q, and B if and only if it satisfies the following constrained system of equations:

$$Mz - w + v + q = 0 \tag{3.1}$$
$$w^T(z - l) = 0 \tag{3.2}$$
$$v^T(u - z) = 0 \tag{3.3}$$
$$z \in B, w \in \Re^n_+, v \in \Re^n_+, \tag{3.4}$$

where $x + \infty = \infty$ for all $x \in \Re$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (3.1) - (3.3) is called a complementary triple.

The objective of the linear model solver is to construct a path from a given complementary triple $(\hat{z}, \hat{w}, \hat{v})$ to a solution $(\bar{z}, \bar{w}, \bar{v})$. The algorithm used to solve the linear problem is identical to that given in [9]; however, artificial variables are incorporated into the model. The augmented system is then:

$$Mz - w + v + Da + \frac{(1-t)}{s}(sr) + q = 0 \tag{3.5}$$
$$w^T(z - l) = 0 \tag{3.6}$$
$$v^T(u - z) = 0 \tag{3.7}$$
$$z \in B, w \in \Re^n_+, v \in \Re^n_+, a \equiv 0, t \in [0, 1] \tag{3.8}$$

where r is the residual, t is the path parameter, and a is a vector of artificial variables. The residual is scaled by s to improve numerical stability.

The addition of artificial variables enables us to construct an initial invertible basis consistent with the given starting point even under rank deficiency. The procedure consists of two parts: constructing an initial guess as to the

basis and then recovering from rank deficiency to obtain an invertible basis. The crash technique gives a good approximation to the active set. The first phase of the algorithm uses this information to construct a basis by partitioning the variables into three sets:

1. $W = \{i \in \{1, \ldots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0\}$
2. $V = \{i \in \{1, \ldots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{v}_i > 0\}$
3. $Z = \{1, \ldots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \ldots, n\}$. Using the above guess, we can recover an invertible basis consistent with the starting point by defining D appropriately. The technique relies upon the factorization to tell the linearly dependent rows and columns of the basis matrix. Some of the variables may be nonbasic, but not at their bounds. For such variables, the corresponding artificial will be basic.

We use a modified version of EXPAND [22] to perform the ratio test. Variables are prioritized as follows:

1. t leaving at its upper bound.
2. Any artificial variable.
3. Any z, w, or v variable.

If a choice as to the leaving variable can be made while maintaining numerical stability and sparsity, we choose the variable with the highest priority (lowest number above).

When an artificial variable leaves the basis and a z-type variable enters, we have the choice of either increasing or decreasing that entering variable because it is nonbasic but not at a bound. The determination is made such that t increases and stability is preserved.

If the code is forced to use a ray start at each iteration (`lemke_start always`), then the code carries out Lemke's method, which is known [7] not to cycle. However, by default, we use a regular start to guarantee that the generated path emanates from the current iterate. Under appropriate conditions, this guarantees a decrease in the nonlinear residual. However, it is then possible for the pivot sequence in the linear model to cycle. To prevent this undesirable outcome, we attempt to detect the formation of a cycle with

the heuristic that if a variable enters the basis more that a given number of times, we are cycling. The number of times the variable has entered is reset whenever t increases beyond its previous maximum or an artificial variable leaves the basis. If cycling is detected, we terminate the linear solver at the largest value of t and return this point.

Another heuristic is added when the linear code terminates on a ray. The returned point in this case is not the base of the ray. We move a slight distance up the ray and return this new point. If we fail to solve the linear subproblem five times in a row, a Lemke ray start will be performed in an attempt to solve the linear subproblem. Computational experience has shown this to be an effective heuristic and generally results in solving the linear model. Using a Lemke ray start is not the default mode, since typically many more pivots are required.

For time when a Lemke start is actually used in the code, an advanced ray can be used. We basically choose the "closest" extreme point of the polytope and choose a ray in the interior of the normal cone at this point. This helps to reduce the number of pivots required. However, this can fail when the basis corresponding to the cell is not invertible. We then revert to the Lemke start.

Since the EXPAND pivot rules are used, some of the variable may be nonbasic, but slightly infeasible, as the solution. Whenever the linear code finisher, the nonbasic variables are put at their bounds and the basic variable are recomputed using the current factorization. This procedure helps to find the best possible solution to the linear system.

The resulting linear solver as modified above is robust and has the desired property that we start from $(\hat{z}, \hat{w}, \hat{v})$ and construct a path to a solution.

3.2.5 Other Features

Some other heuristics are incorporated into the code. During the first iteration, if the linear solver fails to find a Newton point, a Lemke start is used. Furthermore, under repeated failures during the linear solve, a Lemke starts will be attempted. A gradient step can also be used when we fail repeatedly.

The proximal perturbation is shrunk each major iteration. However, when numerical difficulties are encountered, it will be increase to a fraction of the current merit function value. These are determined as when the linear solver returns the Reset or Singular status.

Spacer steps are taken every major iteration, in which the iterate is chosen

to be the best point for the normal map. The corresponding basis passed into the Lemke code is also updated.

Scaling is done based on the diagonal of the matrix passed into the linear solver.

We finally note, that we the merit function fails to show sufficient decrease over the last 100 iterates, a restart will be performed, as this indicates we are close to a stationary point.

3.3 Difficult Models

3.3.1 Ill-Defined Models

A problem can be ill-defined for several different reasons. We concentrate on the following particular cases. We will call F well-defined at $\bar{x} \in C$ if $\bar{x} \in D$ and ill-defined at $\bar{x}$ otherwise. Furthermore, we define F to be well-defined near $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $C \cap \mathcal{N}(\bar{x}) \subseteq D$. By saying the function is well-defined near $\bar{x}$, we are simply stating that F is defined for all $x \in C$ sufficiently close to $\bar{x}$. A function not well-defined near $\bar{x}$ is termed ill-defined near $\bar{x}$.

We will say that F has a well-defined Jacobian at $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $\mathcal{N}(\bar{x}) \subseteq D$ and F is continuously differentiable on $\mathcal{N}(\bar{x})$. Otherwise the function has an ill-defined Jacobian at $\bar{x}$. We note that a well-defined Jacobian at $\bar{x}$ implies that the MCP has a well-defined function near $\bar{x}$, but the converse is not true.

PATH uses both function and Jacobian information in its attempt to solve the MCP. Therefore, both of these definitions are relevant. We discuss cases where the function and Jacobian are ill-defined in the next two subsections. We illustrate uses for the merit function information and final point statistics within the context of these problems.

Function Undefined

We begin with a one-dimensional problem for which F is ill-defined at $x = 0$ as follows:

$$0 \leq x \quad \perp \quad \tfrac{1}{x} \geq 0.$$

Here x must be strictly positive because $\frac{1}{x}$ is undefined at $x = 0$. This condition implies that $F(x)$ must be equal to zero. Since $F(x)$ is strictly

```
positive variable x;
equations F;

F.. 1 / x =g= 0;

model simple / F.x /;

x.l = 1e-6;

solve simple using mcp;
```

Figure 3.1: GAMS Code for Ill-Defined Function

positive for all x strictly positive, this problem has no solution.

We are able to perform this analysis because the dimension of the problem is small. Preprocessing linear problems can be done by the solver in an attempt to detect obviously inconsistent problems, reduce problem size, and identify active components at the solution. Similar processing can be done for nonlinear models, but the analysis becomes more difficult to perform. Currently, PATH only checks the consistency of the bounds and removes fixed variables and the corresponding complementary equations from the model.

A modeler might not know a priori that a problem has no solution and might attempt to formulate and solve it. GAMS code for this model is provided in Figure 3.1. We must specify an initial value for x in the code. If we were to not provide one, GAMS would use $x = 0$ as the default value, notice that F is undefined at the initial point, and terminate before giving the problem to PATH. The error message problem indicates that the function $\frac{1}{x}$ is ill-defined at $x = 0$, but does not determine whether the corresponding MCP problem has a solution.

After setting the starting point, GAMS generates the model, and PATH proceeds to "solve" it. A portion of the output relating statistics about the solution is given in Figure 3.2. PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in Section 3.2.1 at the final point. The Normal Map merit function, and to a lesser extent, the complementarity error, indicate that the "solution" found does not necessarily solve the MCP.

To indicate the difference between the merit functions, Figure 3.3 plots them all for the simple example. We note that as x approaches positive infin-

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e+00 eqn: (F)
Inf-Norm of Normal Map. . . . .  1.1181e+16 eqn: (F)
Inf-Norm of Minimum Map . . . .  8.9441e-17 eqn: (F)
Inf-Norm of Fischer Function. .  8.9441e-17 eqn: (F)
Inf-Norm of Grad Fischer Fcn. .  8.9441e-17 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . .  8.9441e-17 var: (X)
Maximum of F. . . . . .  1.1181e+16 eqn: (F)
Maximum of Grad F . . . . . .  1.2501e+32 eqn: (F)
                                          var: (X)
```

Figure 3.2: PATH Output for Ill-Defined Function

ity, numerically, we are at a solution to the problem with respect to all of the merit functions except for the complementarity error, which remains equal to one. As x approaches zero, the merit functions diverge, also indicating that $x = 0$ is not a solution.

The natural residual and Fischer function tend toward 0 as $x \downarrow 0$. From these measures, we might think $x = 0$ is the solution. However, as previously remarked F is ill-defined at $x = 0$. F and ∇F become very large, indicating that the function (and Jacobian) might not be well-defined. We might be tempted to conclude that if one of the merit function indicators is not close to zero, then we have not found a solution. This conclusion is not always the case. When one of the indicators is non-zero, we have reservations about the solution, but we cannot eliminate the possibility that we are actually close to a solution. If we slightly perturb the original problem to

$$0 \leq x \quad \perp \quad \tfrac{1}{x+\epsilon} \geq 0$$

for a fixed $\epsilon > 0$, the function is well-defined over $C = \mathbf{R}^n_+$ and has a unique solution at $x = 0$. In this case, by starting at $x > 0$ and sufficiently small, all of the merit functions, with the exception of the Normal Map, indicate that we have solved the problem as is shown by the output in Figure 3.4 for $\epsilon = 1{*}10^{-6}$ and $x = 1{*}10^{-20}$. In this case, the Normal Map is quite large and we might think that the function and Jacobian are undefined. When only the normal map is non-zero, we may have just mis-identified the optimal basis. By setting the `merit_function normal` option, we can resolve the problem, identify the correct basis, and solve the problem with all indicators being

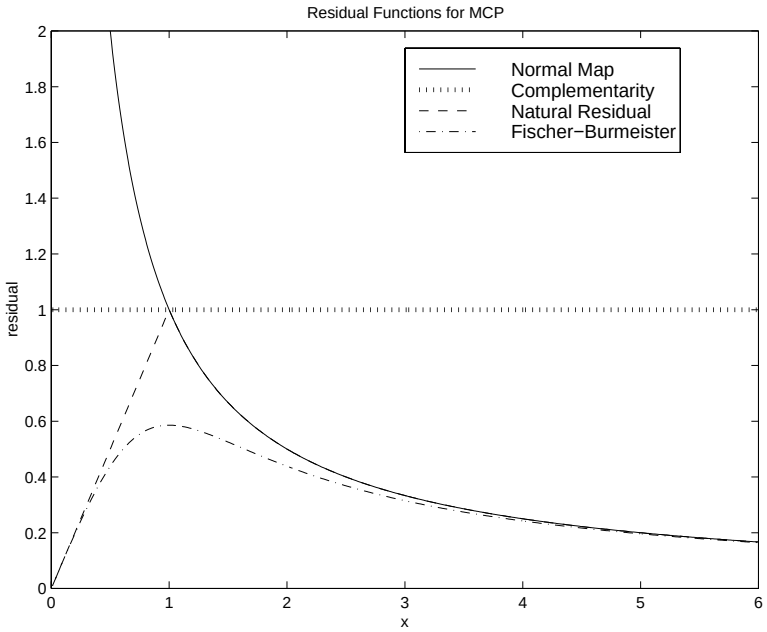

Figure 3.3: Merit Function Plot

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e-14 eqn: (G)
Inf-Norm of Normal Map. . . . .  1.0000e+06 eqn: (G)
Inf-Norm of Minimum Map . . . .  1.0000e-20 eqn: (G)
Inf-Norm of Fischer Function. .  1.0000e-20 eqn: (G)
Inf-Norm of Grad Fischer Fcn. .  1.0000e-20 eqn: (G)

FINAL POINT STATISTICS
Maximum of X. . . . . .  1.0000e-20 var: (X)
Maximum of F. . . . . .  1.0000e+06 eqn: (G)
Maximum of Grad F . . . . . .  1.0000e+12 eqn: (G)
                                          var: (X)
```

Figure 3.4: PATH Output for Well-Defined Function

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e-07 eqn: (F)
Inf-Norm of Normal Map. . . . .  1.0000e-07 eqn: (F)
Inf-Norm of Minimum Map . . . .  1.0000e-07 eqn: (F)
Inf-Norm of Fischer Function. .  2.0000e-07 eqn: (F)
Inf-Norm of Grad FB Function. .  2.0000e+00 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . .  1.0000e-14 var: (X)
Maximum of F. . . . . .  1.0000e-07 eqn: (F)
Maximum of Grad F . . . . . .  5.0000e+06 eqn: (F)
                                          var: (X)
```

Figure 3.5: PATH Output for Ill-Defined Jacobian

close to zero. This example illustrates the point that all of these tests are not infallible. The modeler still needs to do some detective work to determine if they have found a solution or if the algorithm is converging to a point where the function is ill-defined.

Jacobian Undefined

Since PATH uses a Newton-like method to solve the problem, it also needs the Jacobian of F to be well-defined. One model for which the function is well-defined over C, but for which the Jacobian is undefined at the solution is: $0 \leq x \perp -\sqrt{x} \geq 0$. This model has a unique solution at $x = 0$.

Using PATH and starting from the point $x = 1 * 10^{-14}$, PATH generates the output given in Figure 3.5. We can see the that gradient of the Fischer Function is nonzero and the Jacobian is beginning to become large. These conditions indicate that the Jacobian is undefined at the solution. It is therefore important for a modeler to inspect the given output to guard against such problems.

If we start from $x = 0$, PATH correctly informs us that we are at the solution. Even though the entries in the Jacobian are undefined at this point, the GAMS interpreter incorrectly returns a value of 0 to PATH. This problem with the Jacobian is therefore undetectable by PATH. (This problem has been fixed in versions of GAMS beyond 19.1).

```
INITIAL POINT STATISTICS
Maximum of X. . . . . .  4.1279e+06 var: (w.29)
Maximum of F. . . . . .  2.2516e+00 eqn: (a1.33)
Maximum of Grad F . . . . . .  6.7753e+06 eqn: (a1.29)
                                          var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . .  9.4504e+06 eqn: (a2.29)
Minimum Row Norm. . . . . .  2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4504e+06 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-03 var: (w.10)
```

Figure 3.6: PATH Output - Poorly Scaled Model

3.3.2 Poorly Scaled Models

Problems which are well-defined can have various numerical problems that can impede the algorithm's convergence. One particular problem is a badly scaled Jacobian. In such cases, we can obtain a poor "Newton" direction because of numerical problems introduced in the linear algebra performed. This problem can also lead the code to a point from which it cannot recover.

The final model given to the solver should be scaled such that we avoid numerical difficulties in the linear algebra. The output provided by PATH can be used to iteratively refine the model so that we eventually end up with a well-scaled problem. We note that we only calculate our scaling statistics at the starting point provided. For nonlinear problems these statistics may not be indicative of the overall scaling of the model. Model specific knowledge is very important when we have a nonlinear problem because it can be used to appropriately scale the model to achieve a desired result.

We look at the `titan.gms` model in MCPLIB, that has some scaling problems. The relevant output from PATH for the original code is given in Figure 3.6. The maximum row norm is defined as

$$\max_{1 \le i \le n} \sum_{1 \le j \le n} \mid (\nabla F(x))_{ij} \mid$$

and the minimum row norm is

$$\min_{1 \le i \le n} \sum_{1 \le j \le n} \mid (\nabla F(x))_{ij} \mid .$$

```
INITIAL POINT STATISTICS
Maximum of X. . . . . .  1.0750e+03 var: (x1.49)
Maximum of F. . . . . .  3.9829e-01 eqn: (g.10)
Maximum of Grad F . . . . . .  6.7753e+03 eqn: (a1.29)
                                          var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . .  9.4524e+03 eqn: (a2.29)
Minimum Row Norm. . . . . .  2.7680e+00 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4904e+03 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-01 var: (w.10)
```

Figure 3.7: PATH Output - Well-Scaled Model

Similar definitions are used for the column norm. The norm numbers for this particular example are not extremely large, but we can nevertheless improve the scaling. We first decided to reduce the magnitude of the `a2` block of equations as indicated by PATH. Using the GAMS modeling language, we can scale particular equations and variables using the .scale attribute. To turn the scaling on for the model we use the .scaleopt model attribute. After scaling the `a2` block, we re-ran PATH and found an additional blocks of equations that also needed scaling, `a2`. We also scaled some of the variables, `g` and `w`. The code added to the model follows:

```
titan.scaleopt = 1;
a1.scale(i) = 1000;
a2.scale(i) = 1000;
g.scale(i) = 1/1000;
w.scale(i) = 100000;
```

After scaling these blocks of equations in the model, we have improved the scaling statistics which are given in Figure 3.7 for the new model. For this particular problem PATH cannot solve the unscaled model, while it can find a solution to the scaled model. Using the scaling language features and the information provided by PATH we are able to remove some of the problem's difficulty and obtain better performance from PATH.

It is possible to get even more information on initial point scaling by inspecting the GAMS listing file. The equation row listing gives the values of all the entries of the Jacobian at the starting point. The row norms generated by PATH give good pointers into this source of information.

```
INITIAL POINT STATISTICS
Zero column of order. . . . . .  0.0000e+00 var: (X)
Zero row of order . . . . . .  0.0000e+00 eqn: (F)
Total zero columns. . . . . .  1
Total zero rows . . . . . .  1
Maximum of F. . . . . .  1.0000e+00 eqn: (F)
Maximum of Grad F . . . . . .  0.0000e+00 eqn: (F)
                                          var: (X)
```

Figure 3.8: PATH Output - Zero Rows and Columns

Not all of the numerical problems are directly attributable to poorly scaled models. Problems for which the Jacobian of the active constraints is singular or nearly singular can also cause numerical difficulty as illustrated next.

3.3.3 Singular Models

Assuming that the problem is well-defined and properly scaled, we can still have a Jacobian for which the active constraints are singular or nearly singular (i.e. it is ill-conditioned). When problems are singular or nearly singular, we are also likely to have numerical problems. As a result the "Newton" direction obtained from the linear problem solver can be very bad. In PATH, we can use proximal perturbation or add artificial variables to attempt to remove the singularity problems from the model. However, it is most often beneficial for solver robustness to remove singularities if possible.

The easiest problems to detect are those for which the Jacobian has zero rows and columns. A simple problem for which we have zero rows and columns is:

$$-2 \leq x \leq 2 \quad \perp \quad -x^2 + 1.$$

Note that the Jacobian, $-2x$, is non-singular at all three solutions, but singular at the point $x = 0$. Output from PATH on this model starting at $x = 0$ is given in Figure 3.8. We display in the code the variables and equations for which the row/column in the Jacobian is close to zero. These situations are problematic and for nonlinear problems likely stem from the modeler providing an inappropriate starting point or fixing some variables resulting in some equations becoming constant. We note that the solver may perform well in the presence of zero rows and/or columns, but the modeler should make sure

that these are what was intended.

Singularities in the model can also be detected by the linear solver. This in itself is a hard problem and prone to error. For matrices which are poorly scaled, we can incorrectly identify "linearly dependent" rows because of numerical problems. Setting `output_factorization_singularities yes` in an options file will inform the user which equations the linear solver thinks are linearly dependent. Typically, singularity does not cause a lot of problems and the algorithm can handle the situation appropriately. However, an excessive number of singularities are cause for concern. A further indication of possible singularities at the solution is the lack of quadratic convergence to the solution.

Appendix A

Case Study: Von Thunen Land Model

We now turn our attention towards using the diagnostic information provided by PATH to improve an actual model. The Von Thunen land model, is a problem renowned in the mathematical programming literature for its computational difficulty. We attempt to understand more carefully the facets of the problem that make it difficult to solve. This will enable to outline and identify these problems and furthermore to extend the model to a more realistic and computationally more tractable form.

A.1 Classical Model

The problem is cast in the Arrow-Debreu framework as an equilibrium problem. The basic model is a closed economy consisting of three economic agents, a landowner, a worker and a porter. There is a central market, around which concentric regions of land are located. Since the produced goods have to be delivered to the market, this is an example of a spatial price equilibrium. The key variables of the model are the prices of commodities, land, labour and transport. Given these prices, it is assumed that the agents demand certain amounts of the commodities, which are supplied so as to maximize profit in each sector. Walras' law is then a consequence of the assumed competitive paradigm, namely that supply will equal demand in the equilibrium state.

We now describe the problems that the consumers and the producers face. We first look at consumption, and derive a demand function for each of the

consumer agents in the economy. Each of these agents has a utility function, that they wish to maximize subject to their budgetary constraints. As is typical in such problems, the utility function is assumed to be Cobb-Douglas

$$u_a(d) = \prod_c d_c^{\alpha_{c,a}}, \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1,$$

where the $\alpha_{c,a}$ are given parameters dependent only on the agent. For each agent a, the variables d_c represent quantities of the desired commodities c. In the Von Thunen model, the goods are wheat, rice, corn and barley. The agents endowments determine their budgetary constraint as follows. Given current market prices, an agents wealth is the value of the initial endowment of goods at those prices. The agents problem is therefore

$$\max_d u_a(d) \text{ subject to } \langle p,\ d\rangle \leq \langle p,\ e_a\rangle, d \geq 0,$$

where e_a is the endowment bundle for agent a. A closed form solution, corresponding to demand from agent a for commodity c is thus

$$d_{c,a}(p) := \frac{\alpha_{c,a} \langle p,\ e_a\rangle}{p_c}.$$

Note that this assumes the prices of the commodities p_c are positive.

The supply side of the economy is similar. The worker earns a wage w_L for his labour input. The land is distributed around the market in rings with a rental rate w_r associated with each ring r of land. The area of land a_r in each ring is an increasing function of r. The model assumes that labour and land are substitutable via a constant elasticities of substitution (CES) function.

Consider the production $x_{c,r}$ of commodity c in region r. In order to maximize profit (or minimize costs), the labour y_L and land use y_r solve

$$\min w_L y_L + w_r y_r \text{ subject to } \phi_c y_L^{\beta_c} y_r^{1-\beta_c} \geq x_{c,r},\ y_L, y_r \geq 0, \tag{A.1}$$

where ϕ_c is a given cost function scale parameter, and $\beta_c \in [0,1]$ is the share parameter. The technology constraint is precisely the CES function allowing a suitable mix of labour and land use. Again, a closed form solution can be calculated. For example, the demand for labour in order to produce $x_{c,r}$ of commodity c in region r is given by

$$x_{c,r} \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L}.$$

Considering all such demands, this clearly assumes the prices of inputs w_L, w_r are positive. A key point to note is that input commodity (factor) demands to produce $x_{c,r}$ can be determined by first solving (A.1) for unit demand $x_{c,r} \equiv 1$ and then multiplying these factor demands by the actual amount desired. Let $\bar{y}_L$ and $\bar{y}_r$ denote the optimal solutions of (A.1) with $x_{c,r} \equiv 1$. Using this fact, the *unit* production cost $\gamma_{c,r}$ for commodity c in region r can be calculated as follows:

$$\begin{aligned}
\gamma_{c,r} &= w_L \bar{y}_L + w_r \bar{y}_r \\
&= w_L \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L} + w_r \frac{(1-\beta_c) \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_r} \\
&= \frac{1}{\phi_c} \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}.
\end{aligned}$$

Transportation is provided by a porter, earning a wage w_p. If we denote the unit cost for transportation of commodity c by t_c, then unit transportation cost to market is

$$T_{c,r}(w_p) := t_c d_r w_p,$$

where d_r is the distance of region r to the market. Spatial price equilibrium arises from the consideration:

$$0 \leq x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \geq p_c.$$

This is intuitively clear; it states that commodity c will be produced in region r only if the combined cost of production and transportation equals the market price.

The above derivations assumed that the producers and consumers acted as price takers. Walras' law is now invoked to determine the prices so that markets clear. The resulting complementarity problem is:

$$\gamma_{c,r} = \frac{1}{\phi_c} \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c} \tag{A.2}$$

$$0 \leq x_{c,r} \quad \perp \quad \gamma_{c,r} + T_{c,r}(w_p) \geq p_c \tag{A.3}$$

$$0 \leq w_L \quad \perp \quad e_L \geq \sum_{r,c} x_{c,r} \frac{\beta_c \gamma_{c,r}}{w_L} \tag{A.4}$$

$$0 \leq w_r \quad \perp \quad a_r \geq \sum_{c} \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r} \tag{A.5}$$

$$0 \leq w_p \quad \perp \quad e_P \geq \sum_{r,c} t_c d_r x_{c,r} \tag{A.6}$$

$$0 \leq p_c \quad \perp \quad \sum_r x_{c,r} \geq \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O} \sum_r w_r a_r}{p_c} \tag{A.7}$$

Note that in (A.4), (A.5) and (A.6), the amounts of labour, land and transport are bounded from above, and hence the prices on these inputs are determined as multipliers (or shadow prices) on the corresponding constraints. The final relationship (A.7) in the above complementarity problem corresponds to market clearance; prices are nonnegative and can only be positive if supply equals demand. (Some modelers multiply the last inequality throughout by p_c. This removes problems where p_c becomes zero, but can also introduce spurious solutions.)

The Arrow-Debreu theory guarantees that the problem is homogeneous in prices; $(x, \lambda w, \lambda p)$ is also a solution whenever (x, w, p) solves the above. Typically this singularity in the model is removed by fixing a numeraire, that is fixing a price (for example $w_L = 1$) and dropping the corresponding complementary relationship.

Unfortunately, in this formulation even after fixing a numeraire, some of the variables p and w may go to zero, resulting in an ill-defined problem. In the case of the Von Thunen land model, the rental price of land w_r decreases as the distance to market increases, and for remote rings of land, it becomes zero. A standard modeling fix is to put artificial lower bounds on these variables. Even with this fix, the problem typically remains very hard to solve. More importantly, the homogeneity property of the prices used above to fix a numeraire no longer holds, and the corresponding complementary relationship (which was dropped from the problem) may fail to be satisfied. It therefore matters which numeriare is fixed, and many modelers run into difficulty since in many cases the solution found by a solver is invalid for the originally posed model.

In order to test our diagnostic information, we implemented a version of the above model in GAMS. The model corresponds closely to the MCPLIB model pgvon105.gms except we added more regions to make the problem even more difficult. The model file has been documented more fully, and the data rounded to improve clarity.

The first trial we attempted was to solve the model without fixing a numeraire. In this case, PATH 4.x failed to find a solution. At the starting point, the indicators described in Section 3.3.1 are reasonable, and there

are no zero rows/columns in the Jacobian. At the best point found, all indicators are still reasonable. However, the listing file indicates a large number of division by zero problems occurring in (A.5). We also note that a nonzero proximal perturbation is used in the first iteration of the crash method. This is an indication of singularities. We therefore added an option to output factorization singularities, and singularities appeared in the first iteration. At this point, we decided to fix a numeraire to see if this alleviated the problem.

We chose to fix the labour wage rate to 1. After increasing the iterations allowed to 100,000, PATH 4.x solved the problem. The statistics at the solution are cause for concern. In particular, the gradient of the Fischer function is 7 orders of magnitude larger than all the other residuals. Furthermore, the Jacobian is very large at the solution point. Looking further in the listing file, a large number of division by zero problems occur in (A.5).

To track down the problem further, we added an artificial lower bound on the variables w_r of 10^{-5}, that would not be active at the aforementioned solution. Resolving gave the same "solution", but resulted in the domain errors disappearing.

Although the problem is solved, there is concern on two fronts. Firstly, the gradient of the Fischer function should go to zero at the solution. Secondly, if a modeler happens to make the artificial lower bounds on the variables a bit larger, then they become active at the solution, and hence the constraint that has been dropped by fixing the price of labour at 1 is violated at this point. Of course, the algorithm is unable to detect this problem, since it is not part of the model that is passed to it, and the corresponding output looks satisfactory.

We are therefore led to the conclusion that the model as postulated is ill-defined. The remainder of this section outlines two possible modeling techniques to overcome the difficulties with ill-defined problems of this type.

A.2 Intervention Pricing

The principal difficulty is the fact that the rental prices on land go to zero as proximity to the market decreases, and become zero for sufficiently remote rings. Such a property is unlikely to hold in a practical setting. Typically, a landowner has a minimum rental price (for example, land in fallow increases in value). As outlined above, a fixed lower bound on the rental price vi-

olates the well-established homogeneity property. A suggestion postulated by Professor Thomas Rutherford is to allow the landowner to intervene and "purchase-back" his land whenever the rental cost gets smaller than a certain fraction of the labour wage.

The new model adds a (homogeneous in price) constraint

$$0 \leq i_r \quad \perp \quad w_r \geq 0.0001 * w_L$$

and modifies (A.5) and (A.7) as follows:

$$\begin{aligned} 0 \leq w_r \quad &\perp \quad a_r - i_r \geq \sum_c \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r} \\ 0 \leq p_c \quad &\perp \quad \sum_r x_{c,r} \geq \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O} \sum_r w_r(a_r - i_r)}{p_c}. \end{aligned} \quad \text{(A.8)}$$

Given the intervention purchase, we can now add a lower bound on w_r to avoid division by zero errors. In our model we chose 10^{-5} since this will never be active at the solution and therefore will not affect the positive homogeneity. After this reformulation, PATH 4.x solves the problem. Furthermore, the gradient of the Fischer function, although slightly larger than the other residuals, is quite small, and can be made even smaller by reducing the convergence tolerance of PATH. Inspecting the listing file, the only difficulties mentioned are division by zero errors in the market clearance condition (A.8), that can be avoided a posteori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

A.3 Nested Production and Maintenance

Another observation that can be used to overcome the land price going to zero is the fact that land typically requires some maintenance labour input to keep it usable for crop growth. Traditionally, in economics, this is carried out by providing a nested CES function as technology input to the model. The idea is that commodity c in region r is made from labour and an intermediate good. The intermediate good is "maintained land". Essentially, the following

production problem replaces (A.1):

$$\begin{array}{cc} \min_{y_M,y_L,y_r,g} & w_L(y_M+y_L)+w_r y_r \\ \text{subject to} & y_r \geq (1-\beta_c-\epsilon)g \\ & y_M \geq \epsilon g \\ & \phi_c y_L^{\beta_c} g^{1-\beta_c} \geq 1, \\ & y_M, y_L, y_r, g \geq 0. \end{array}$$

Note that the variable y_M represents "maintenance labour" and g represents the amount of "maintained land" produced, an intermediate good. The process of generating maintained land uses a Leontieff production function, namely

$$\min(\lambda_r y_r, \lambda_M y_M) \geq g.$$

Here $\lambda_M = \frac{1}{\epsilon}$, ϵ small, corresponds to small amounts of maintenance labour, while $\lambda_r = \frac{1}{1-\beta_c-\epsilon}$ is chosen to calibrate the model correctly. A simple calculus exercise then generates appropriate demand and cost expressions. The resulting complementarity problem comprises (A.3), (A.6), (A.7) and

$$\begin{aligned} \gamma_{c,r} &= \frac{w_L^{\beta_c}}{\phi_c}\left(\frac{w_L\epsilon + w_r(1-\beta_c-\epsilon)}{1-\beta_c}\right)^{1-\beta_c} \\ 0 \leq w_L \quad \perp \quad e_L &\geq \sum_{r,c} x_{c,r}\gamma_{c,r}\left(\frac{\beta_c}{w_L} + \frac{\epsilon(1-\beta_c)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)}\right) \\ 0 \leq w_r \quad \perp \quad a_r &\geq \sum_c \frac{x_{c,r}\gamma_{c,r}(1-\beta_c)(1-\beta_c-\epsilon)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)} \end{aligned}$$

After making the appropriate modifications to the model file, PATH 4.x solved the problem on defaults without any difficulties. All indicators showed the problem and solution found to be well-posed.

Bibliography

[1] S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.

[2] S. C. Billups. Improving the robustness of descent-based mehtods for semi-smooth equations using proximal perturbations. *Mathematical Programming*, 87:153–176, 2000.

[3] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76:533–562, 1997.

[4] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[5] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Study*, 16:1–17, 1982.

[6] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[7] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[8] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, Boston, 1992.

[9] S. P. Dirkse. *Robust Solution of Mixed Complementarity Problems*. PhD thesis, Computer Sciences Department, University of Wisconsin,

Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[10] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[11] S. P. Dirkse and M. C. Ferris. A pathsearch damped Newton method for computing general equilibria. *Annals of Operations Research*, pages 211–232, 1996.

[12] S. P. Dirkse and M. C. Ferris. Crash techniques for large-scale complementarity problems. In Ferris and Pang [19], pages 40–61.

[13] S. P. Dirkse and M. C. Ferris. Traffic modeling and variational inequalities using GAMS. In Ph. L. Toint, M. Labbe, K. Tanczos, and G. Laporte, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, volume 166 of *NATO ASI Series F*, pages 136–163. Springer-Verlag, 1998.

[14] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86:475–497, 1999.

[15] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81:53–71, 1994.

[16] M. C. Ferris, A. Meeraus, and T. F. Rutherford. Computing Wardropian equilibrium in a complementarity framework. *Optimization Methods and Software*, 10:669–685, 1999.

[17] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12:207–227, 1999.

[18] M. C. Ferris and T. S. Munson. Preprocessing complementarity problems. Mathematical Programming Technical Report 99-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.

[19] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*, Philadelphia, Pennsylvania, 1997. SIAM Publications.

[20] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39:669–713, 1997.

[21] A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.

[22] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[23] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23:707–716, 1986.

[24] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59:779–805, 1991.

[25] P. T. Harker and J. S. Pang. Finite–dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48:161–220, 1990.

[26] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *The Economic Journal*, 107:1405–1430, 1997.

[27] J. Huang and J. S. Pang. Option pricing and linear complementarity. *Journal of Computational Finance*, 2:31–60, 1998.

[28] N. H. Josephy. Newton's method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.

[29] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[30] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley and Los Angeles, 1951.

[31] C. E. Lemke and J. T. Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12:413–423, 1964.

[32] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.

[33] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37:1–18, 1987.

[34] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.

[35] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19:1299–1324, 1995.

[36] P. Tseng. Growth behavior of a class of merit functions for the nonlinear complementarity problem. *Journal of Optimization Theory and Applications*, 89:17–37, 1996.

[37] S. J. Wright. *Primal–Dual Interior–Point Methods*. SIAM, Philadelphia, Pennsylvania, 1997.

GAMS/SBB

Contents

Release Notes

- April 30, 2002: Level 009
  - NLP solvers sometimes have difficulty proving the optimality of a good point. The way they report that solution is with solver status "Terminated by Solver" and model status "Intermediate Nonoptimal". SBB's default behavior is to ignore such a solution (potentially go into failseq). With the new option `acceptnonopt` these solutions are accepted for further action.
  - SBB offers node selections that switch between DFS and best bound/best estimate selection. In DFS mode SBB usually switches back to best bound/estimate if the DFS search resulted in a pruned or infeasible node or a new integer solution was found. In these cases it can be advantageous to search the close neighborhood of that node also in a DFS fashion. With the new option `dfsstay` SBB is instructed to do some more DFS nodes even after a switch to best bound/estimate has been requested.
- January 11, 2002: Level 008
  - Maintenance release
- December 11, 2001: Level 007
  - NLP solvers sometimes have difficulty solving particular nodes and using up all the resources in this node. SBB provides options (see `subres, subiter`) to overcome these instances, but these options must be set in advance. SBB now keeps track of how much time is spent in the nodes, builds an average over time, and automatically controls the time spend in each node. The option `avgresmult` allows the user to customize this new feature.
- November 13, 2001: Level 006
  - Maintenance release
- May 22, 2001: Level 005
  - SBB derives an implicit, absolute termination tolerance if the model has a `discrete` objective row. This may speed up the overall time if the user has tight termination tolerances (optca, optcr).

  - SBB passes indices of rows with domain violations back to the LST file. All domain violation from the root node and from all sub nodes are reported, and the user can take advantage of this information to overcome these violations.

- March 21, 2001: Level 004
  - Pseudo Costs are available in SBB. Check section SBB with Pseudo Costs
  - A mix of DFS/Best Bound/Best Estimate node selection schemes are available through the `nodesel` option.
  - The `tryint` option now works for integer variables.
  - Additional information about node and variable selection can be printed to the Log file using the `printbbinfo` option.
- December 22, 2000: Level 003
  - MINOS and SNOPT are available as NLP subsolvers.
- November 19, 2000: Level 002
  - The model and solver status returned by SBB are synchronized with return codes reported by DICOPT.
- October 16, 2000: Level 001
  - SBB introduced to GAMS distribution 19.5.
  - CONOPT and CONOPT2 are the available NLP subsolvers.

1 Introduction

SBB is a new GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use

- CONOPT
- MINOS
- SNOPT

as solvers for submodels.

SBB supports all types of discrete variables supported by GAMS, including:

- Binary
- Integer
- Semicont
- Semiint
- Sos1
- Sos2

2 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option `infeasseq` and `failseq` below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the

NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option `infeasseq` below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

3 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh, A Computational Study of Search Strategies for Mixed Integer Programming, INFORMS Journal on Computing, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option `varsel`). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option `nodesel`).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are **not** good candidates for search strategies which require pseudo costs (`varsel 3, nodesel 3,5,6`). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (`varsel 3, nodesel 3,5,6`) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

4 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are `iterlim, reslim, nodlim, optca, optcr, optfile, cheat, cutoff, prioropt`, and `tryint`. A description of all available GAMS options can be found in Chapter "Using Solver Specific Options".

If you specify "<`modelname`>`.optfile = 1;`" before the SOLVE statement in your GAMS model, SBB will then look for and read an option file with the name *sbb.opt* (see "Using Solver Specific Options" for general use of solver option files). Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```
optname value
```

with one option on each line.

For example,

```
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file *conopt.opt* will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:

Option	Description	Default
rootsolver	solver[.n] Solver is the name of the GAMS NLP solver that should be used in the root node, and n is the integer corresponding to optfile for the root node. If .n is missing, the optfile treated as zero (i.e., the NLP solver) will not look for an options file. This SBB option can be used to overwrite the default that uses the NLP solver specified with an Option NLP = *solver*; statement or the default GAMS solver for NLP.	GAMS NLP solver
subsolver	solver[.n] Similar to rootsolver but applied to the subnodes.	GAMS NLP solver
loginterval	The interval (number of nodes) for which log lines are written.	1
loglevel	The level of log output: 0: only SBB log lines with one line every loginterval nodes 1: NLP solver log for the root node plus SBB loglines as 0 2: NLP solver log for all nodes plus SBB log lines as 0	1
subres	The default for subres passed on through reslim. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.	reslim
subiter	The default for subiter passed on through iterlim. Similar to subres but sets the iteration limit for solving a node in the B&B tree.	iterlim
failseq	solver1[.n1] solver2[.n2] ... where solver1 is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain, limit and does not return a locally optimal or locally infeasible solution. n1 is the value of optfile passed to the alternative NLP solver. If .n1 is left blank it is interpreted as zero. Similarly, solver2 is the name of a GAMS NLP solver that is used if solver1 fails, and n2 is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more solver.n pairs. You can use the same solver several times with different options files. failseq conopt conopt.2 conopt.3 means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file *conopt.op2*, and if it again fails, try CONOPT with options file *conopt.op3*. If all solver and options file combinations fail the node will be labeled "ignored" and the node will not be explored further. The default is to try only one solver (the rootsolver or subsolver) and to ignore nodes with a solver failure.	None
infeasseq	level solver1[.n1] solver2[.n2] ... The purpose of infeasseq is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node "Locally Infeasible" and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value <level>. If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that Locally Infeasible nodes are indeed infeasible and to remove them from further consideration.	None
acceptnonopt	If this option is set to 1 and the subsolver terminates with solver status "Terminated by Solver" and model status "Intermediate Nonoptimal" SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the failseq is consulted.	0

Option	Description	Default
avgresmult	Similar to subres, this options allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor avgresmult is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the failseq are called. The default multiplier avgresmult is 5. Setting avgresmult to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at least 5 seconds to solve the node. The competing option subres overwrites the automatically generated resource limit.	5
nodesel	Node selection: 0: automatic 1: Depth First Search (DFS) 2: Best Bound (BB) 3: Best Estimate (BE) 4: DFS/BB mix 5: DFS/BE mix 6: DFS/BB/BE mix	0
dfsstay	If the node selection is a B*/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B* node selction mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting dfsstay to n instructs SBB to stay in DFS mode for another n nodes.	0
varsel	Variable selection: 0: automatic 1: maximum integer infeasibility 2: minimum integer infeasibility 3: pseudo costs	0
epint	The integer infeasibility tolerance.	1.0e-5
memnodes	The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.	10000
printbbinfo	Additional info of log output: 0: no additional info 1: the node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.	0
intsollim	maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.	99999

5 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the loginterval and loglevel options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

```
Root node solved locally optimal.
   Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
      0     0   0    8457.6878   3            -       8457.6878       -
      1     1   1    8491.2869   2            -       8457.6878       -
      2     2   2    8518.1779   1            -       8457.6878       -
*     3     3   3    9338.1020   0    9338.1020       8457.6878  0.1041
```

```
     4     2   1       pruned   -    9338.1020       8491.2869  0.0997
Solution satisfies optcr
Statistics:
   Iterations    :                 90
   NLP Seconds   :           0.110000
   B&B nodes     :                  3
   MIP solution  :        9338.101979 found in node 3
   Best possible :        8491.286941
   Absolute gap  :         846.815039     optca :  0.000000
   Relative gap  :           0.099728     optcr :  0.100000
   Model Status  :                  8
   Solver Status :                  1

NLP Solver Statistics
   Total Number of NLP solves  :           7
   Total Number of NLP failures:           0
   Details:         conopt
     # execs             7
     # failures          0
Terminating.
```

The fields in the log are:

Field	Description
`Node`	The number of the current node. The root node is node 0.
`Act`	The number of active nodes defined as the number of subnodes that have not yet been solved.
`Lev`	The level in the search tree, i.e., the number of branches needed to reach this node.
`Objective`	The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. "pruned" indicates that the objective value was worse than the Best Integer value, "infeasible" indicates that the node was Infeasible or Locally Infeasible, and "ignored" indicates that the node could not be solved (see under failseq above).
`IInf`	The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to `IInf`.
`Best Int`	The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (*) in column one indicates that the node is integer and that the solution is better than the best yet found.
`Best Bound`	The minimum value of "Objective" for the subnodes hat have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.
`Gap`	The relative gap between the Best Integer solution and the Best Bound.

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text "ignored" in the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
   1     1   1   infeasible   -            -       6046.0186       -
   2     0   1    6042.0995  10            -       6042.0995       -
```

```
   3     1   2      ignored   -            -       6042.0995       - (4,6)
   4     0   2    5804.5856   8            -       5804.5856       -
   5     1   3      ignored   -            -       5804.5856       - (4,7)
```

The next Log file shows the effect of the `infeasseq` and `failseq` options on the model above. CONOPT with options file *conopt.opt* (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT1, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT fails but CONOPT1 finds a Locally Optimal solution, and this solution is then used for further search. The option file for the following run would be:

```
rootsolver conopt.1
subsolver conopt.1
infeasseq conopt1 minos snopt
```

The log looks as follows:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
conopt.1 reports locally infeasible
Executing conopt1
conopt1 reports locally infeasible
Executing minos
minos reports locally infeasible
Executing snopt
   1     1    1  infeasible   -            -       6046.0186       -
   2     0    1   6042.0995  10            -       6042.0995       -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 INTERMEDIATE NONOPTIMAL
Executing conopt1
   3     1    2   4790.2373   8            -       6042.0995       -
   4     2    3   4481.4156   6            -       6042.0995       -
conopt.1 reports locally infeasible
Executing conopt1
conopt1 reports locally infeasible
Executing minos
minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE
Executing snopt
   5     3    4  infeasible   -            -       6042.0995       -
   6     2    4   4480.3778   4            -       6042.0995       -
```

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

```
NLP Solver Statistics
   Total Number of NLP solves  :   45
   Total Number of NLP failures:   13
   Details:     conopt     minos   snopt
     # execs        34         3       8
     # failures      4         3       6
```

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These *jumps* of the objective in the *wrong* direction which might also have an impact on the best bound/possible are reported in a separate statistic:

```
Non convex model!
   # jumps in best bound       :           2
   Maximum jump in best bound  :   20.626587 in node 13
   # jumps to better objective :           2
   Maximum jump in objective   :   20.626587 in node 13
```

6 Comparison of DICOPT and SBB

Until recently, MINLP models could only be solved with the DICOPT solver. DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using cheap linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.

SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored.

Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).

GAMS/SNOPT: AN SQP ALGORITHM FOR LARGE-SCALE CONSTRAINED OPTIMIZATION

PHILIP E. GILL* WALTER MURRAY†
MICHAEL A. SAUNDERS† ARNE DRUD‡
ERWIN KALVELAGEN§

January 19, 2000

1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

$$\begin{array}{|lll|}\hline \text{NP} & \underset{x}{\text{minimize or maximize}}\ f(x) & \\ & F(x) \sim b_1 & \\ & \text{subject to} \quad Gx \sim b_2 & \\ & l \leq x \leq u, & \\ \hline \end{array} \tag{1}$$

where $x \in \Re^n$, $f(x)$ is a linear or nonlinear smooth objective function, l and u are constant lower and upper bounds, $F(x)$ is a set of nonlinear constraint functions, G is a sparse matrix, $\sim$ is a vector of relational operators ($\leq$, $\geq$ or $=$), and b_1 and b_2 are right-hand side constants. $F(x) \sim b_1$ are the nonlinear constraints of the model and $Gx \sim b_2$ form the linear constraints.

The gradients of f and F_i are automatically provided by GAMS, using its automatic differentiation engine.

*Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112.
†Department of EESOR, Stanford University, Stanford, CA 94305-4023
‡ARKI Consulting and Development, Bagsvaerd, Denmark
§GAMS Development Corp., Washington DC

The bounds may include special values `-INF` or `+INF` to indicate $l_j = -\infty$ or $u_j = +\infty$ for appropriate j. Free variables have both bounds infinite and fixed variables have $l_j = u_j$.

1.1 Problem Types

If the nonlinear functions are absent, the problem is a *linear program* (LP) and SNOPT applies the primal simplex method [2]. Sparse basis factors are maintained by LUSOL [11] as in MINOS [14].

If only the objective is nonlinear, the problem is *linearly constrained* (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [6], using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [7, 9].

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [12, 13]. It is suitable for nonlinear problems with thousands of constraints and variables, but not thousands of degrees of freedom. (Thus, for large problems there should be many constraints and bounds, and many of them should be active at a solution.)

1.2 Selecting the SNOPT Solver

The GAMS system can be instructed to use the SNOPT solver by incorporating the following option in the GAMS model:

```
option NLP=SNOPT;
```

If the model contains non-smooth functions like abs(x), or max(x, y) you can try to get it solved by SNOPT using

```
option DNLP=SNOPT;
```

These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they are not too close to an optimum.

It is also possible to specify `NLP=SNOPT` or `DNLP=SNOPT` on the command line, as in:

```
> gamslib chem
> gams chem nlp=snopt
```

2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described in [6].

2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
obj.. z =e= sum(i, sqr(resid(i)));

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize z subject to $z = \sum_i resid_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize $\sum_i resid_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model `chem.gms` from the model library solves in 16 iterations. When we add the bound

```
energy.lo = 0;
```

on the objective variable `energy` and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
variables x,y,z;
equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
```

```
model m /all/;
option nlp=snopt;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\text{minimize } f(x) = (1+x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see §4.1).

2.2 Constraints and slack variables

The m general constraints of the problem (1) are formed by $F(x) \sim b_1$ and $Gx \sim b_2$. SNOPT will add to each general constraint a slack variable s_i with appropriate bounds. The problem defined by (1) can therefore be rewritten in the following equivalent form:

$$\begin{aligned} &\underset{x,s}{\text{minimize}} \quad f(x) \\ &\text{subject to } \begin{pmatrix} F(x) \\ Gx \end{pmatrix} - s = 0, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u. \end{aligned}$$

where a maximization problem is cast into a minimization by multiplying the objective function by -1.

The linear and nonlinear general constraints become equalities of the form $F(x) - s_N = 0$ and $Gx - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

2.3 Major iterations

The basic structure of SNOPT's solution algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate $\{x_k\}$ a QP subproblem is used to generate a search direction towards the next iterate $\{x_{k+1}\}$. The constraints of the subproblem are formed from the linear constraints $Gx - s_L = 0$ and the nonlinear constraint linearization

$$F(x_k) + F'(x_k)(x - x_k) - s_N = 0,$$

where $F'(x_k)$ denotes the *Jacobian*: a matrix whose rows are the first derivatives of $F(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{aligned} F'(x_k)x \quad -s_N \qquad &= -F(x_k) + F'(x_k)x_k, \\ Gx \qquad\quad -s_L &= 0, \end{aligned}$$

where x and s are bounded by l and u as before. If the $m \times n$ matrix A and m-vector b are defined as

$$A = \begin{pmatrix} F'(x_k) \\ G \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -F(x_k) + F'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} \;\; q(x) \quad \text{subject to} \;\; Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \tag{2}$$

where $q(x)$ is a quadratic approximation to a modified Lagrangian function [6].

2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure, with the *minor* iterations of an SQP method being the iterations of the QP method. At each minor iteration, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B, x_S and x_N are called the *basic*, *superbasic* and *nonbasic* variables respectively; they are a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will be equal to one of their upper or lower bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or the sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero at a solution for LP problems.

If it appears that no improvement can be made with the current definition of B, S and N, a nonbasic variable is selected to be added to S, and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints in $Ax - s = b$ are the *dual variables* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j. The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics. In practice, an *approximate* QP solution is found by relaxing these conditions on d_j (see the `Minor optimality tolerance` in §4.3).

2.5 The merit function

After a QP subproblem has been solved, new estimates of the NP solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T\big(F(x) - s_N\big) + \tfrac{1}{2}\big(F(x) - s_N\big)^T D\big(F(x) - s_N\big), \quad (3)$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ denotes the optimal QP solution, the linesearch determines a step α_k $(0 < \alpha_k \leq 1)$ such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \widehat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix} \quad (4)$$

gives a *sufficient decrease* in the merit function. When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [9]. As in NPSOL, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [7, 3].

2.6 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

FLP	$\underset{x,v,w}{\text{minimize}}$	$e^T(v + w)$
	subject to	$Gx - v + w \sim b_2, \quad l \leq x \leq u, \quad v, w \geq 0,$

where e is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called elastic programming. We also describe it as minimizing the ℓ_1 norm of the infeasibilities.)

If the linear constraints are infeasible ($v \neq 0$ or $w \neq 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and solves the problem

NP(γ)	$\underset{x,v,w}{\text{minimize}}$	$f(x) + \gamma e^T(v + w)$
	subject to	$\begin{pmatrix} F(x) - v + w \\ Gx \end{pmatrix} \sim b, \quad l \leq x \leq u, \quad v, w \geq 0,$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x)+\gamma e^T(v+w)$ is called a *composite objective*. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the Sℓ_1QP algorithm of Fletcher [4]. See also Conn [1].

3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of "restart" information in case a number of related models is solved in a row. Starting from an optimal point from a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the `.L` and `.M` fields of variables and equations.

3.1 Starting points

To specify a starting point for SNOPT use the `.L` level values in GAMS. For example, to set all variables $x_{i,j} := 1$ use `x.l(i,j)=1;`. The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i):

Example	$\underset{r,a,b}{\text{minimize}} \quad r$
	subject to $\quad (x_i-a)^2+(y_i-b)^2 \le r^2, \quad r \ge 0.$

This problem can be modeled in GAMS as follows.

```
set i points /p1*p10/;

parameters
    x(i)    x coordinates,
    y(i)    y coordinates;

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a       x coordinate of center of circle
    b       y coordinate of center of circle
    r       radius;

equations
    e(i)    points must be inside circle;
```

```
e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;
```

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:

```
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement `Y.LO = 1;`, the `SOLVE` statement will override the default level of zero of such a variable in order to make it feasible.

Note: another way to formulate the model would be to minimize r^2 instead of r. This allows SNOPT to solve the problem even with the default starting point.

3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the `chem.gms` model from the model library:

```
model mixer chemical mix for N2H4+O2 / all /;

solve mixer minimizing energy using nlp;
solve mixer minimizing energy using nlp;
```

we observe the following log:

```
[erwin@hamilton]$ gams chem nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                          G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chem.gms(48) 1 Mb
--- Starting execution
--- chem.gms(42) 1 Mb
--- Generating model MIXER
--- chem.gms(46) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj   Objective   Optimal   nS PD
     0     5 0.0E+00     1  3.292476E+01 2.1E-01    0 TF   r
     1     4 1.0E+00     2  4.517191E+01 2.2E-01    1 TF n r
     2    10 8.6E-02     5  4.533775E+01 1.7E-01    6 TF s
     3     3 1.0E+00     6  4.608439E+01 7.6E-02    6 TF
     4     2 1.0E+00     7  4.667813E+01 1.4E-01    5 TF
     5     2 1.0E+00     8  4.751149E+01 2.2E-02    6 TF
     6     3 1.0E+00     9  4.757024E+01 2.1E-02    4 TF
     7     2 7.1E-01    11  4.763634E+01 3.3E-02    5 TF
     8     3 1.0E+00    12  4.768395E+01 3.3E-02    7 TF
     9     3 4.6E-01    14  4.769958E+01 1.9E-02    5 TF
    10     2 1.0E+00    15  4.770539E+01 1.5E-02    6 TF

 Major Minor   Step   nObj   Objective   Optimal   nS PD
    11     1 1.0E+00    16  4.770639E+01 6.2E-04    6 TF
    12     1 1.0E+00    17  4.770650E+01 5.0E-03    6 TF
    13     1 1.0E+00    18  4.770651E+01 1.6E-04    6 TF
    14     1 1.0E+00    19  4.770651E+01 1.8E-05    6 TF
    15     1 1.0E+00    20  4.770651E+01 2.7E-05    6 TF
    16     1 1.0E+00    21  4.770651E+01 7.6E-07    6 TT

 EXIT - Optimal Solution found.
```

```
--- Restarting execution
--- chem.gms(46) 1 Mb
--- Reading solution for model MIXER
--- chem.gms(46) 1 Mb
--- Generating model MIXER
--- chem.gms(47) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj   Objective   Optimal   nS PD
     0     0 0.0E+00     1  4.770651E+01 7.4E-07    0 TT   r

 EXIT - Optimal Solution found.

--- Restarting execution
--- chem.gms(47) 1 Mb
--- Reading solution for model MIXER
--- chem.gms(47) 1 Mb
*** Status: Normal completion
[erwin@hamilton]$
```

The first `solve` takes 16 iterations, while the second `solve` needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

`X.M` $= 0$	basic
`X.M` $\neq 0$	nonbasic if level value is at bound, superbasic otherwise

A marginal value of `EPS` means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the `.M` values. It is further noted that if too many `.M` values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the `bratio` option (see §4.1).

4 Options

In many cases NLP models can be solved with GAMS/SNOPT without using solver options. For special situations it is possible to specify non-standard values for some or all of the options.

4.1 GAMS options

The following GAMS options affect the behavior of SNOPT.

NLP

This option Selects the NLP solver. Example: `option NLP=SNOPT;`. See also §1.2.

DNLP

Selects the DNLP solver. Example: `option DNLP=SNOPT;`. See also §1.2.

iterlim

Sets the (minor) iteration limit. Example: `option iterlim=50000;`. The default is 10000. SNOPT will stop as soon as the number of *minor iterations* exceeds the iteration limit. In that case the current solution will be reported.

reslim

Sets the time limit or resource limit. Depending on the architecture this is wall clock time or CPU time. SNOPT will stop as soon as more than *reslim* seconds have elapsed since SNOPT started. The current solution will be reported in this case. Example: `option reslim = 600;`. The default is 1000 seconds.

domlim

Sets the domain violation limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating x^y for $x < 0$ (x^y is evaluated as $\exp(y \log x)$). When such a situation occurs the number of domain errors is increased by one, and SNOPT will stop if this number exceeds the limit. If the limit has not been reached, a reasonable number is returned (e.g., in the case of $\sqrt{x}, x < 0$ a zero is passed back) and SNOPT is asked to continue. In many cases SNOPT will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`. Example: `option domlim=100;`. The default value is 0.

bratio

Basis acceptance test. When several models are solved in a row, GAMS automatically passes dual information to SNOPT so that it can reconstruct an advanced basis. When too many new variables or constraints enter the model, it may be better not to use existing basis information, but to *crash* a new basis instead. The `bratio` determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis. Example: `option bratio=1.0;`. Default: bratio = 0.25.

sysout

Debug listing. When turned on, extra information printed by SNOPT

will be added to the listing file. Example: `option sysout=on;`. Default: sysout = off.

work
: The `work` option sets the amount of memory SNOPT can use. By default an estimate is used based on the model statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes etc.). In most cases this is sufficient to solve the model. In some extreme cases SNOPT may need more memory, and the user can specify this with this option. For historical reasons `work` is specified in "double words" or 8 byte quantities. For example, `option work=100000;` will ask for 0.76 MB (a megabyte being defined as 1024×1024 bytes).

reform
: This option will instruct the reformulation mechanism described in §2.1 to substitute out equality equations. The default value of 100 will cause the procedure to try further substitutions along the diagonal after the objective variable has been removed. Any other value will prohibit this diagonal procedure. Example: `option reform = 0;`. Default: reform = 100.

4.2 Model suffices

A model identifier in GAMS can have several suffices to which you can assign values. A small GAMS fragment illustrates this:

```
model m /all/;
m.iterlim = 3000;
solve m minimizing z using nlp;
```

Options set by assigning to the suffixed model identifier override the global options. For example,

```
model m /all/;
m.iterlim = 3000;
option iterlim = 100;
solve m minimizing z using nlp;
```

will use an iteration limit of 3000 for this solve.

m.iterlim
: Sets the iteration limit. Overrides the global iteration limit. Example: `m.iterlim=50000;` The default is 10000. See also §4.1.

m.reslim
: Sets the resource or time limit. Overrides the global resource limit. Example: `m.reslim=600;` The default is 1000 seconds. See also §4.1.

m.bratio

Sets the basis acceptance test parameter. Overrides the global setting. Example: `m.bratio=1.0;` The default is 0.25. See also §4.1.

m.scaleopt

Whether or not to scale the model using user-supplied scale factors. The user can provide scale factors using the `.scale` variable and equation suffix. For example, `x.scale(i,j) = 100;` will assign a scale factor of 100 to all $x_{i,j}$ variables. The variables SNOPT will see are scaled by a factor $1/variable\_scale$, so the modeler should use scale factors that represent the order of magnitude of the variable. In that case SNOPT will see variables that are scaled around 1.0. Similarly equation scales can be assigned to equations, which are scaled by a factor $1/equation\_scale$. Example: `m.scaleopt=1;` will turn scaling on. The default is not to use scaling, and the default scale factors are 1.0. Automatic scaling is provided by the SNOPT option `scale option`.

m.optfile

Sets whether or not to use a solver option file. Solver specific SNOPT options are specified in a file called `snopt.opt`, see §4.3. To tell SNOPT to use this file, add the statement: `option m.optfile=1;`. The default is not to use an option file.

m.workspace

The workspace option sets the amount of memory that SNOPT can use. By default an estimate is used based on the model statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc.). In most cases this is sufficient to solve the model. In some extreme cases SNOPT may need more memory, and the user can specify this with this option. The amount of memory is specified in MB. Example: `m.workspace = 5;`.

4.3 SNOPT options

SNOPT options are specified in a file called `snopt.opt` which should reside in the working directory (this is the project directory when running models from the IDE). To tell SNOPT to read this file, use the statement `m.optfile = 1;` in the model (see §4.2).

An example of a valid `snopt.opt` is:

```
Hessian full memory
Hessian frequency 20
```

Users familiar with the SNOPT distribution from Stanford University will notice that the `begin` and `end` keywords are missing. These markers are optional in GAMS/SNOPT and will be ignored. Therefore, the following option file is also accepted:

```
begin
Hessian full memory
Hessian frequency 20
end
```

All options are case-insensitive. A line is a comment line if it starts with an asterisk, `*`, in column one.

Here follows a description of all SNOPT options that are possibly useful in a GAMS environment:

Check frequency i

Every ith minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.
`Check frequency 1` is useful for debugging purposes, but otherwise this option should not be needed.
Default: `check frequency 60`.

Crash option i

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix $(A \quad -I)$. The `Crash option` i determines which rows and columns of A are eligible initially, and how many times CRASH is called. Columns of $-I$ are used to pad the basis where necessary.

`Crash option 0:` The initial basis contains only slack variables: $B = I$.

`Crash option 1:` CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A.

`Crash option 2:` CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).

`Crash option 3:` CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows before the first major iteration.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to

a bound.) CRASH then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.
Default: `Crash option 3` for linearly constrained problems and `Crash option 0;` for problems with nonlinear constraints.

Crash tolerance r

The `Crash tolerance` r allows the starting procedure CRASH to ignore certain "small" nonzeros in each column of A. If $a_{\max}$ is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)
When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.
For example, suppose the first m columns of A are the matrix shown under `LU factor tolerance`; i.e., a tridiagonal matrix with entries -1, 4, -1. To help CRASH choose all m columns for the initial basis, we would specify `Crash tolerance` r for some value of $r > 1/4$.
Default: `Crash tolerance 0.1`

Elastic weight ω

This parameter denoted by ω determines the initial weight γ associated with problem NP(γ).
At any given major iteration k, elastic mode is started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where g is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.
Thereafter, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP(γ), but not feasible for NP. After the rth increase, $\gamma = \omega 10^r(1 + \|g(x_{k1})\|_2)$, where x_{k1} is the iterate at which γ was first needed.
Default: `Elastic weight 10000.0`

Expand frequency i

This option is part of the EXPAND anti-cycling procedure [8] designed to make progress even on highly degenerate problems.
For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the `Minor feasibility tolerance` is δ. Over a period of i iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (in steps of $0.5\delta/i$).
For nonlinear models, the same procedure is used for iterations in which

there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.
Increasing the expand frequency helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see `Pivot tolerance`).
Default: `Expand frequency 10000`

Factorization frequency k

At most k basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the `Check frequency`) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

Default: `Factorization frequency 100` for linear programs and `Factorization frequency 50` for nonlinear models.

Feasibility tolerance t

See `Minor feasibility tolerance`.
Default: `Feasibility tolerance 1.0e-6`

Feasible point only

This options means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.
Default: turned off.

Feasible exit

If SNOPT is about to terminate with nonlinear constraint violations, the option `Feasible exit` requests further effort to satisfy the nonlinear constraints while ignoring the objective function.
Default: turned off.

Hessian Full memory

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major

iteration.)
If `Hessian Full memory` is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect 1-step Q-superlinear convergence to the solution.
Default: turned on when the number of nonlinear variables $n_1 \leq 75$.

Hessian Limited memory
This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)
`Hessian Limited memory` should be used on problems where the number of nonlinear variables n_1 is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)
Default: turned on when the number of nonlinear variables $n_1 > 75$.

Hessian frequency i
If `Hessian Full` is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)
`Hessian Full memory` and `Hessian frequency = 20` have a similar effect to `Hessian Limited memory` and `Hessian updates = 20` (except that the latter retains the current diagonal during resets).
Default: `Hessian frequency 99999999` (i.e. never).

Hessian updates i
If `Hessian Limited memory` is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.
Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).
Default: `Hessian updates 20` (only when limited memory storage model is used).

Infeasible exit ok
If SNOPT is about to terminate with nonlinear constraint violations, the companion option `Feasible exit` requests further effort to satisfy the nonlinear constraints while ignoring the objective function. `Infeasible`

`exit ok` does not do this extra effort.
Default: turned on.

Iterations limit k
This is the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option overrides the GAMS iterlim options.
Default: specified by GAMS.

Linesearch tolerance t
This controls the accuracy with which a steplength will be located along the direction of search each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated.

- t must be a real value in the range $0.0 \leq t \leq 1.0$.
- The default value $t = 0.9$ requests just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate (this is usually the case for GAMS models), a more accurate search may be appropriate; try $t = 0.1$, 0.01 or 0.001. The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try $t = 0.99$. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)

Default: `Linesearch tolerance 0.9`.

Log frequency k
See `Print frequency`.
Default: `Log frequency 1`

LU factor tolerance r_1
LU update tolerance r_2
These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy r_1, $r_2 \geq 1.0$. The matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix},$$

where the multipliers μ satisfy $|\mu| \leq r_i$. Smaller values of r_i favor stability, while larger values favor sparsity.

- For large and relatively dense problems, $r_1 = 5.0$ (say) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a submatrix of the form

$$\begin{pmatrix} 4 & -1 & & & & & \\ -1 & 4 & -1 & & & & \\ & -1 & 4 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 4 & -1 & \\ & & & & -1 & 4 & \end{pmatrix},$$

  both r_1 and r_2 should be in the range $1.0 \leq r_i < 4.0$.

Defaults for linear models: `LU factor tolerance 100.0` and `LU update tolerance 10.0`.
The defaults for nonlinear models are `LU factor tolerance 5.0` and `LU update tolerance 5.0`.

LU density tolerance r_1

The density tolerance r_1 is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1, the Markowitz strategy for choosing pivots is terminated. The remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.
Default: `LU density tolerance 0.6`

LU singularity tolerance r_2

The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)
In some cases, the Jacobian may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting a larger tolerance r_2 = `1.0e-5`, say, may help cause a judicious change of basis.
Default: `LU singularity tolerance 3.7e-11` for most machines. This value corresponds to $\epsilon^{2/3}$, where ϵ is the relative machine precision.

Major feasibility tolerance ϵ_r

This specifies how accurately the nonlinear constraints should be satisfied. The default value of `1.0e-6` is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let `rowerr` be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\texttt{rowerr} = \max_i \ \texttt{viol}_i/\|x\| \ \leq \ \epsilon_r, \tag{5}$$

where $\texttt{viol}_i$ is the violation of the ith nonlinear constraint ($i = 1 : \texttt{nnCon}$, `nnCon` being the number of nonlinear constraints).

In the GAMS/SNOPT iteration log, `rowerr` appears as the quantity labeled "`Feasibl`". If some of the problem functions are known to be of low accuracy, a larger `Major feasibility tolerance` may be appropriate.

Default: `Major feasibility tolerance 1.0e-6`.

Major optimality tolerance ϵ_d

This specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\texttt{maxgap} = \max_j \ \texttt{gap}_j/\|\pi\| \ \leq \ \epsilon_d, \tag{6}$$

where $\texttt{gap}_j$ is an estimate of the complementarity gap for variable j ($j = 1 : n + m$). The gaps are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $(A \quad -I)$, and π is the set of QP dual variables):

$$\texttt{gap}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, `maxgap` appears as the quantity labeled "`Optimal`".

Default: `Major optimality tolerance 1.0e-6`.

Major iterations limit k

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints.

Default: `Major iterations limit` $\max\{1000, 3\max\{m, n\}\}$.

Major print level p

This controls the amount of output to the GAMS listing file each major iteration. This output is only visible if the `sysout` option is turned on (see §4.1). `Major print level 1` gives normal output for linear and nonlinear problems, and `Major print level 11` gives additional details of the Jacobian factorization that commences each major iteration.

In general, the value being specified may be thought of as a binary number of the form

`Major print level JFDXbs`

where each letter stands for a digit that is either `0` or `1` as follows:

- `s` a single line that gives a summary of each major iteration. (This entry in `JFDXbs` is not strictly binary since the summary line is printed whenever `JFDXbs` ≥ 1).
- `b` BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if `JFDXbs` ≥ 10).
- `X` x_k, the nonlinear variables involved in the objective function or the constraints.
- `D` π_k, the dual variables for the nonlinear constraints.
- `F` $F(x_k)$, the values of the nonlinear constraint functions.
- `J` $J(x_k)$, the Jacobian.

To obtain output of any items `JFDXbs`, set the corresponding digit to `1`, otherwise to `0`.
If `J=1`, the Jacobian will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J=1`, there is no reason to specify `X=1` unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3  1.250000D+01 BS      1  1.00000E+00      4  2.00000E+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.
`Major print level 0` suppresses most output, except for error messages.
Default: `Major print level 00001`

Major step limit r

This parameter limits the change in x during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1+\|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.

3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The `Major step limit` provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: `Major step limit 2.0`.

Minor iterations limit k

This is the maximum number of minor iterations allowed for each QP subproblem in the SQP algorithm. Current experience is that the major iterations converge more reliably if the QP subproblems are allowed to solve accurately. Thus, k should be a large value.
In the major iteration log, a `t` at the end of a line indicates that the corresponding QP was terminated by the limit k.
Note that the SNOPT option `Iterations limit` or the GAMS `iterlim` option defines an independent limit on the *total* number of minor iterations (summed over all QP subproblems).
Default: `Minor iterations limit` $\max\{1000, 5\max\{m, n\}\}$

Minor feasibility tolerance t

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within the tolerance t. This includes slack variables. Hence, general linear constraints should also be satisfied to within t.
Feasibility with respect to nonlinear constraints is judged by the `Major feasibility tolerance` (not by t).

- If the bounds and linear constraints cannot be satisfied to within t, the problem is declared *infeasible*. Let `sInf` be the corresponding sum of infeasibilities. If `sInf` is quite small, it may be appropriate to raise t by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t =$ 1.0e-6, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)
- If `Scale option` ≥ 1, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).

- In reality, SNOPT uses t as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the `Elastic` options.

Default: `Minor feasibility tolerance 1.0e-6`.

Minor optimality tolerance t

This is used to judge optimality for each QP subproblem. Let the QP reduced gradients be $d_j = g_j - \pi^T a_j$, where g_j is the jth component of the QP gradient, a_j is the associated column of the QP constraint matrix, and π is the set of QP dual variables.

- By construction, the reduced gradients for basic variables are always zero. The QP subproblem will be declared optimal if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy

$$d_j/\|\pi\| \geq -t \quad \text{or} \quad d_j/\|\pi\| \leq t$$

  respectively, and if $|d_j|/\|\pi\| \leq t$ for superbasic variables.

- In the above tests, $\|\pi\|$ is a measure of the size of the dual variables. It is included to make the tests independent of a large scale factor on the objective function. The quantity actually used is defined by

$$\|\pi\| = \max\{\sigma/\sqrt{m}, 1\}, \quad \text{where} \quad \sigma = \sum_{i=1}^{m} |\pi_i|.$$

- If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against t.

Default: `Minor optimality tolerance 1.0e-6`.

Minor print level k

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. This option is only useful if the `sysout` option is turned on (see §4.1). The value of k has the following effect:

0	No minor iteration output except error messages.
≥ 1	A single line of output each minor iteration (controlled by `Print frequency`).
≥ 10	Basis factorization statistics generated during the periodic refactorization of the basis (see `Factorization frequency`). Statistics for the *first factorization* each major iteration are controlled by the `Major print level`.

Default: `Minor print level 0`.

Optimality tolerance t

See `Minor optimality tolerance`.
Default: `Optimality tolerance 1.0e-6`.

Partial Price i

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (when a nonbasic variable is selected to become superbasic).

- When $i = 1$, all columns of the constraint matrix $(A \quad -I)$ are searched.
- Otherwise, A and I are partitioned to give i roughly equal segments A_j, I_j ($j = 1$ to i). If the previous pricing search was successful on A_j, I_j, the next search begins on the segments A_{j+1}, I_{j+1}. (All subscripts here are modulo i.)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2}, and so on.
- `Partial price` t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

Default: `Partial price 10` for linear models and `Partial price 1` for nonlinear models.

Pivot tolerance r During solution of QP subproblems, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When x changes to $x + \alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.
- Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the `Minor Feasibility tolerance` (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.
- To a lesser extent, the `Expand frequency` (say f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

Default: `Pivot tolerance 3.7e-11` on most machines. This corresponds to $\epsilon^{2/3}$ where ϵ is the machine precision.

Print frequency k

When `sysout` is turned on (see §4.1) and `Minor print level` ≥ 1, a line of the QP iteration log will be printed on the listing file every kth minor iteration.
Default: `Print frequency 1`.

Scale option i

Three scale options are available as follows:

`Scale option 0:` No scaling. This is recommended if it is known that x and the constraint matrix (and Jacobian) never have very large elements (say, larger than 1000).

`Scale option 1:` Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [5]). This will sometimes improve the performance of the solution procedures.

`Scale option 2:` All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A \quad -I)$ that are fixed or have positive lower bounds or negative upper bounds.

If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. `Scale option 2` should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

Default: `Scale option 2` for linear models and `Scale option 1` for NLP's.

Scale tolerance r

This parameter affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made.
Default: `Scale tolerance 0.9`.

Scale Print

This option causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j-n)$ if $j > n$.
The listing file will only show these values if the `sysout` option is turned on (see §4.1).
Default: turned off.

Solution Yes

This option causes the SNOPT solution file to be printed to the GAMS listing file. It is only visible if the `sysout` option is turned on (see §4.1). Default: turned off.

Superbasics limit i

This places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than m, the number of general constraints.) The default value of i is therefore 1.

For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".

Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1. This will save storage if n_1 is very large.

Unbounded objective value $f_{\max}$
Unbounded step size $\alpha_{\max}$

These parameters are intended to detect unboundedness in nonlinear problems. (They may not achieve that purpose!) During a line search, f is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. if $|f|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, iterations are terminated with the exit message `Problem is unbounded (or badly scaled)`.

In a GAMS environment no floating-point overflow errors should occur when singularities are present during the evaluation of $f(x + \alpha p)$ before the test can be made.

Defaults: `Unbounded objective value 1.0e+15` and `Unbounded step size 1.0e+18`.

Violation limit τ

This keyword defines an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of τ. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then τ may be any large

positive value.
Default: `Violation limit 10`.

5 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

```
[erwin@hamilton]$ gamslib chem
Model chem.gms retrieved
[erwin@hamilton]$ gams chem nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                        G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chem.gms(47) 1 Mb
--- Starting execution
--- chem.gms(42) 1 Mb
--- Generating model MIXER
--- chem.gms(46) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj   Objective   Optimal   nS PD
     0     5 0.0E+00     1  3.292476E+01 2.1E-01    0 TF   r
     1     4 1.0E+00     2  4.517191E+01 2.2E-01    1 TF n r
     2    10 8.6E-02     5  4.533775E+01 1.7E-01    6 TF s
     3     3 1.0E+00     6  4.608439E+01 7.6E-02    6 TF
     4     2 1.0E+00     7  4.667813E+01 1.4E-01    5 TF
     5     2 1.0E+00     8  4.751149E+01 2.2E-02    6 TF
     6     3 1.0E+00     9  4.757024E+01 2.1E-02    4 TF
     7     2 7.1E-01    11  4.763634E+01 3.3E-02    5 TF
     8     3 1.0E+00    12  4.768395E+01 3.3E-02    7 TF
     9     3 4.6E-01    14  4.769958E+01 1.9E-02    5 TF
    10     2 1.0E+00    15  4.770539E+01 1.5E-02    6 TF

 Major Minor   Step   nObj   Objective   Optimal   nS PD
    11     1 1.0E+00    16  4.770639E+01 6.2E-04    6 TF
    12     1 1.0E+00    17  4.770650E+01 5.0E-03    6 TF
    13     1 1.0E+00    18  4.770651E+01 1.6E-04    6 TF
    14     1 1.0E+00    19  4.770651E+01 1.8E-05    6 TF
    15     1 1.0E+00    20  4.770651E+01 2.7E-05    6 TF
    16     1 1.0E+00    21  4.770651E+01 7.6E-07    6 TT

 EXIT - Optimal Solution found.

--- Restarting execution
--- chem.gms(46) 1 Mb
--- Reading solution for model MIXER
```

```
--- chem.gms(46) 1 Mb
*** Status: Normal completion
[erwin@hamilton]$
```

For a nonlinearly constrained problem, the log is somewhat different:

```
[erwin@hamilton]$ gamslib chenery
Model chenery.gms retrieved
[erwin@hamilton]$ gams chenery nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                          G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chenery.gms(240) 1 Mb
--- Starting execution
--- chenery.gms(222) 1 Mb
--- Generating model CHENRAD
--- chenery.gms(225) 1 Mb
---    39 rows, 44 columns, and 133 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.07 Mb

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
     0    39 0.0E+00     1 -1.653933E+07 1.4E+00 1.4E+00    6 0.0E+00 FF   r i
     1     6 1.0E+00     2  6.215366E+03 9.8E-01 1.4E+00    6 0.0E+00 FF n r
     2     2 1.0E+00     3 -4.424844E+03 5.6E-01 7.8E-01    7 2.8E-01 FF s
     3     1 1.0E+00     4  2.756620E+02 1.1E-01 2.1E-01    7 2.8E-01 FF
     4     1 1.0E+00     6  5.640617E+02 5.6E-03 2.2E-01    7 2.8E-01 FF  m
     5     6 1.0E+00     8  6.188177E+02 9.9E-03 2.6E-01    5 2.8E-01 FF  m
     6     2 7.2E-01    11  6.827737E+02 2.7E-02 2.0E-01    4 2.8E-01 FF  m
     7     4 5.9E-01    14  7.516259E+02 3.4E-02 9.4E-02    6 2.8E-01 FF  m
     8     1 1.0E+00    15  8.437315E+02 6.7E-03 1.7E+00    6 2.8E-01 FF
     9     3 1.0E+00    17  8.756771E+02 7.1E-03 3.5E-01    4 2.8E-01 FF  m
    10     5 3.1E-01    21  9.010440E+02 2.4E-02 1.1E+00    6 2.8E-01 FF  m

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    11     2 2.6E-01    24  9.168958E+02 2.5E-02 6.9E-01    5 2.8E-01 FF
    12     1 4.8E-01    26  9.404851E+02 2.9E-02 5.0E-01    5 2.8E-01 FF
    13     3 1.0E+00    27  9.983802E+02 1.6E-02 1.3E-01    6 2.8E-01 FF
    14     1 1.0E+00    28  1.013533E+03 6.2E-04 4.8E-02    6 2.8E-01 FF
    15     2 1.0E+00    29  1.021295E+03 1.1E-02 1.2E-02    5 2.8E-01 FF
    16     1 1.0E+00    30  1.032156E+03 5.2E-03 1.1E-02    5 2.8E-01 FF
    17     2 1.0E+00    31  1.033938E+03 6.7E-05 1.4E-02    4 2.8E-01 FF
    18     2 1.0E+00    32  1.036764E+03 4.5E-04 1.0E-02    3 2.8E-01 FF
    19     2 1.0E+00    33  1.037592E+03 6.5E-05 3.0E-02    2 2.8E-01 FF
    20     1 1.0E+00    34  1.039922E+03 4.6E-04 4.4E-02    2 2.8E-01 FF
    21     2 1.0E+00    35  1.040566E+03 1.4E-05 7.8E-02    3 2.8E-01 FF

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    22     4 1.0E+00    36  1.056256E+03 1.3E-02 5.6E-02    4 2.8E-01 FF
    23     2 1.0E+00    37  1.053213E+03 1.9E-04 3.1E-03    3 3.9E-01 FF
    24     2 1.0E+00    38  1.053464E+03 2.2E-05 4.7E-03    2 3.9E-01 FF
    25     1 1.0E+00    39  1.053811E+03 3.7E-05 1.8E-02    2 3.9E-01 FF
```

```
    26     1 1.0E+00    40  1.055352E+03 1.1E-03 3.9E-02    2 3.9E-01 FF
    27     1 1.0E+00    41  1.055950E+03 1.3E-03 1.5E-02    2 3.9E-01 FF
    28     1 1.0E+00    42  1.056047E+03 1.5E-06 1.3E-02    2 3.9E-01 FF
    29     1 1.0E+00    43  1.056991E+03 3.2E-04 2.2E-02    2 3.9E-01 FF
    30     1 1.0E+00    44  1.058439E+03 2.7E-03 1.8E-02    2 3.9E-01 FF
    31     3 1.0E+00    45  1.058885E+03 2.2E-04 9.3E-03    2 3.9E-01 FF
    32     1 1.0E+00    46  1.058918E+03 3.3E-05 1.6E-03    2 3.9E-01 FF

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    33     1 1.0E+00    47  1.058920E+03 4.6E-06 1.1E-05    2 3.9E-01 FF
    34     1 1.0E+00    48  1.058920E+03 5.1E-10 5.3E-07    2 3.9E-01 TT

 EXIT - Optimal Solution found.

--- Restarting execution
--- chenery.gms(225) 1 Mb
--- Reading solution for model CHENRAD
--- chenery.gms(239) 1 Mb
*** Status: Normal completion
[erwin@hamilton]$
```

GAMS prints the number of equations, variables and non-zero elements of the model it generated. This gives an indication of the size of the model. SNOPT then says how much memory it allocated to solve the model, based on an estimate. If the user had specified a different amount using the `work` option or the `workspace` model suffix, there would be a message like

```
Work space requested by user   --    0.76 Mb
Work space requested by solver --    0.02 Mb
```

The SNOPT log shows the following columns:

Major The current major iteration number.

Minor is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minor` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see §2).

Step The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

nObj The number of times the nonlinear objective function has been evaluated. `nObj` is printed as a guide to the amount of work required for the linesearch.

nCon The number of times SNOPT evaluated the nonlinear constraint functions.

Merit is the value of the augmented Lagrangian merit function (3). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §2). As the solution is approached, `Merit` will

converge to the value of the objective at the solution.
In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.
If the constraints are linear, this item is labeled `Objective`, the value of the objective function. It will decrease monotonically to its optimal value.

Feasibl is the value of `rowerr`, the maximum component of the scaled nonlinear constraint residual (5). The solution is regarded as acceptably feasible if `Feasibl` is less than the `Major feasibility tolerance`.
If the constraints are linear, all iterates are feasible and this entry is not printed.

Optimal is the value of `maxgap`, the maximum complementarity gap (6). It is an estimate of the degree of nonoptimality of the reduced costs. Both `Feasibl` and `Optimal` are small in the neighborhood of a solution.

nS The current number of superbasic variables.

Penalty is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

PD is a two-letter indication of the status of the convergence tests involving primal and dual feasibility of the iterates (see (5) and (6) in the description of `Major feasibility tolerance` and `Major optimality tolerance`). Each letter is `T` if the test is satisfied, and `F` otherwise.
If either of the indicators is `F` when SNOPT terminates with `0 EXIT -- optimal solution found`, the user should check the solution carefully.

The summary line may include additional code characters that indicate what happened during the course of the iteration.

`c` Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.

`d` During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of `Violation limit`.

`l` The norm-wise change in the variables was limited by the value of the `Major step limit`. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of `Major step limit`.

`i` If SNOPT is not in elastic mode, an "`i`" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP(γ).
If SNOPT is already in elastic mode, an "`i`" indicates that the minimizer

of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`M` An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

`m` This is the same as "`M`" except that it was also necessary to modify the update to include an augmented Lagrangian term.

`R` The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the `Hessian frequency` and `Hessian updates` keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`r` The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`s` A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

`S` This is the same as a "`s`" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

`n` No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

`t` The minor iterations were terminated at the `Minor iteration limit`.

`u` The QP subproblem was unbounded.

`w` A weak solution of the QP subproblem was found.

Finally SNOPT prints an exit message. See §5.1.

5.1 EXIT conditions

When the solution procedure terminates, an `EXIT --` message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

`0 EXIT -- optimal solution found`

This is the message we all hope to see! It is certainly preferable to every other message, and we naturally want to believe what it says, because this is surely one situation where *the computer knows best*.

In all cases, a distinct level of caution is in order, even if it can wait until next morning. For example, if the objective value is much better than expected, we may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other caution about "`Optimal solution`"s. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. `Max Primal infeas` refers to the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller `Minor feasibility tolerance` (say 10 times smaller) and perhaps `Scale option 0`.

Similarly, `Max Dual infeas` indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\text{Max Dual infeas/Norm of pi} = 10^{-d},$$

then the objective function would probably change in the dth significant digit if optimization could be continued. If d seems too large, consider restarting with smaller `Major` and `Minor optimality tolerance`s.

Finally, `Nonlinear constraint violn` shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller `Major feasibility tolerance`.

`1 EXIT -- the problem is infeasible`

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the `Minor feasibility tolerance` are ignored, but at least one component of x or s violates a bound by more than the tolerance.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the `Elastic weight` parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

```
 2  EXIT -- the problem is unbounded  (or badly scaled)
    EXIT -- violation limit exceeded -- the problem may be unbounded
```

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. A message prior to the EXIT message will give the index of the nonbasic variable. Consider adding an upper or lower bound to the variable. Also, examine the constraints that have nonzeros in the associated column, to see if they have been formulated as intended.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the `Scale` option.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the `Unbounded` parameters—see §4), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not

bounded below in the feasible region defined by expanding the bounds by the value of the `Violation limit`.

`3 EXIT -- major iteration limit exceeded`
`EXIT -- minor iteration limit exceeded`
`EXIT -- too many iterations`

Either the `Iterations limit` or the `Major iterations limit` was exceeded before the required solution could be found. Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

`4 EXIT -- requested accuracy could not be achieved`

A feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within 10^{-2} of satisfying the `Major optimality tolerance`. Check that the `Major optimality tolerance` is not too small.

`5 EXIT -- the superbasics limit is too small: nnn`

The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already `nnn` superbasics (and no room for any more).

In general, raise the `Superbasics limit` s by a reasonable amount, bearing in mind the storage needed for the reduced Hessian (about $\frac{1}{2}s^2$ double words).

`6 EXIT -- constraint and objective values could not be calculated`

This exit should not occur in a GAMS environment.

`7 EXIT -- subroutine funobj seems to be giving incorrect gradients`

This exit should not occur in a GAMS environment.

`8 EXIT -- subroutine funcon seems to be giving incorrect gradients`

This exit should not occur in a GAMS environment.

`9 EXIT -- the current point cannot be improved upon`

The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the `optimality tolerance` will probably make this message go away.

`10 EXIT -- cannot satisfy the general constraints`

An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

This probably means that the current basis is very ill-conditioned. Try `Scale option 1` if scaling has not yet been used and there are some linear constraints and variables.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Try setting the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

`12 EXIT -- terminated from subroutine s1User`
This message appears when the *resource limit* was reached (see §4.1) or when the solver was interrupted by a Ctrl-C keyboard signal.

`20 EXIT -- not enough integer/real storage for the basis factors`
Increase the workspace by using the `work` option or the `workspace` model suffix.

`21 EXIT -- error in basis package`
A preceding message will describe the error in more detail. This error should not happen easily in a GAMS environment.

`22 EXIT -- singular basis after nnn factorization attempts`
This exit is highly unlikely to occur. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix U were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized, but singularity persists. This must mean that the problem is badly scaled, or the `LU factor tolerance` is too much larger than 1.0.

`30 EXIT -- the basis file dimensions do not match this problem`
This exit should not occur in a GAMS environment.

`31 EXIT -- the basis file state vector does not match this problem`
This exit should not occur in a GAMS environment.

`32 EXIT -- system error. Wrong no. of basic variables: nnn`
This exit should never happen, and may indicate a configuration problem with your GAMS/SNOPT version.

`42 EXIT -- not enough 8-character storage to start solving the problem`
Increase the workspace by using the `work` option or the `workspace` model suffix.

`43 EXIT -- not enough integer storage to start solving the problem`
Increase the work space by using the `work` option or the `workspace` model suffix.

`44 EXIT -- not enough real storage to start solving the problem`
Increase the work space by using the `work` option or the `workspace` model suffix.

`45 EXIT -- Function evaluation error limit exceeded`
The `function evaluation error limit` was reached. When evaluating a nonlinear function either in the objective or in the constraints, an evaluation error occurred and the allowed number of such errors was exceeded. Either increase the `domlim` option (see §4.1) or preferably add bounds or linear constraints such that these errors cannot happen. The errors are most often caused by declaring x to be a free variable while the model contains functions like $\sqrt{x}$ or $\log(x)$. Overflows in exponentiation x^y are also a common cause for this exit. Inspect the listing file for messages like

```
**** ERROR(S) IN EQUATION PRODF
     2 INSTANCES OF - UNDEFINED LOG OPERATION (RETURNED -0.1E5)
```

The equation name mentioned in this message gives a good indication where to look in the model.

6 Listing file messages

The listing file (`.lst` file) also contains feedback on how the SNOPT solver performed on a particular model. For the `chem.gms` model, the solve summary looks like the following:

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

     MODEL   MIXER               OBJECTIVE  ENERGY
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  SNOPT               FROM LINE  46

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE              -47.7065

 RESOURCE USAGE, LIMIT          0.010     1000.000
 ITERATION COUNT, LIMIT        45        10000
 EVALUATION ERRORS              0            0

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University


 Work space allocated           --    0.02 Mb

 EXIT - Optimal Solution found.

 Major, Minor Iterations    16      45
 Funobj, Funcon calls       22       0
 Superbasics                 6
 Aggregations                0
 Interpreter Usage        0.01   100.0%

 Work space used by solver      --    0.02 Mb
```

The solver completed normally at a local (or possibly global) optimum. A complete list of possible solver status and model status values is in Tables 1 and 2.

The resource usage (time used), iteration count and evaluation errors during nonlinear function and gradient evaluation are all within their limits. These limits can be increased by the option `reslim`, `iterlim` and `domlim` (see §4.1).

The possible `EXIT` messages are listed in §5.1.

The statistics following the `EXIT` message are as follows.

Model status	Remarks
1 OPTIMAL	Applies only to linear models.
2 LOCALLY OPTIMAL	A local optimum in an NLP was found. It may or may not be a global optimum.
3 UNBOUNDED	For LP's this message is reliable. A badly scaled NLP can also cause this message to appear.
4 INFEASIBLE	Applies to LP's: the model is infeasible.
5 LOCALLY INFEASIBLE	Applies to NLP's: Given the starting point, no feasible solution could be found although feasible points may exist.
6 INTERMEDIATE INFEASIBLE	The search was stopped (e.g., because of an iteration or time limit) and the current point violates some constraints or bounds.
7 INTERMEDIATE NONOPTIMAL	The search was stopped (e.g., because of an iteration or time limit) and the current point is feasible but violates the optimality conditions.
8 INTEGER SOLUTION	Does not apply to SNOPT.
9 INTERMEDIATE NON-INTEGER	Does not apply to SNOPT.
10 INTEGER INFEASIBLE	Does not apply to SNOPT.
ERROR UKNOWN	Check listing file for error messages.
ERROR NO SOLUTION	Check listing file for error messages.

Table 1: Model status values

Solver status	Remarks
1 NORMAL COMPLETION	SNOPT completed the optimization task.
2 ITERATION INTERRUPT	Iteration limit was hit. Increase the `iterlim` option (see §4.1).
3 RESOURCE INTERRUPT	Time limit was hit. Increase the `reslim` option (see §4.1).
4 TERMINATED BY SOLVER	Check the listing file.
5 EVALUATION ERROR LIMIT	`domlim` error limit was exceeded. See §4.1.
6 UNKNOWN ERROR	Check the listing file for error messages.
ERROR SETUP FAILURE	Id.
ERROR SOLVER FAILURE	Id.
ERROR INTERNAL SOLVER FAILURE	Id.
ERROR SYSTEM FAILURE	Id.

Table 2: Solver status values

Major, minor iterations. The number of major and minor iterations for this optimization task. Note that the number of minor iterations is the same as reported by `ITERATION COUNT`.

Funobj, Funcon calls. The number of times SNOPT evaluated the objective function $f(x)$ or the constraint functions $F_i(x)$ and their gradients. For a linearly constrained problem the number of `funcon` calls should be zero.

Superbasics. This is number of superbasic variables in the reported solution. See §2.4.

Aggregations. The number of equations removed from the model by the objective function recovery algorithm (see §2.1).

Interpreter usage. This line refers to how much time was spent evaluating functions and gradients. Due to the low resolution of the clock and the small size of this model, it was concluded that 100% of the time was spent inside the routines that do function and gradient evaluations. For larger models these numbers are more accurate.

References

[1] A. R. Conn, *Constrained optimization using a nondifferentiable penalty function*, SIAM J. Numer. Anal., 10 (1973), pp. 760–779.

[2] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[3] S. K. Eldersveld, *Large-scale sequential quadratic programming algorithms*, PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.

[4] R. Fletcher, *An ℓ_1 penalty method for nonlinear constraints*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., Philadelphia, 1985, SIAM, pp. 26–40.

[5] R. Fourer, *Solving staircase linear programs by the simplex method. 1: Inversion*, Math. Prog., 23 (1982), pp. 274–313.

[6] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[7] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *User's guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming*, Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA, 1986.

[8] ———, *A practical anti-cycling procedure for linearly constrained optimization*, Math. Prog., 45 (1989), pp. 437–474.

[9] ———, *Some theoretical properties of an augmented Lagrangian merit function*, in Advances in Optimization and Parallel Computing, P. M. Pardalos, ed., North Holland, North Holland, 1992, pp. 101–128.

[10] ———, *Sparse matrix methods in optimization*, SIAM J. on Scientific and Statistical Computing, 5 (1984), pp. 562–589.

[11] ———, *Maintaining LU factors of a general sparse matrix*, Linear Algebra and its Applications, 88/89 (1987), pp. 239–270.

[12] B. A. Murtagh and M. A. Saunders, *Large-scale linearly constrained optimization*, Math. Prog., 14 (1978), pp. 41–72.

[13] ———, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Prog. Study, 16 (1982), pp. 84–117.

[14] ———, *MINOS 5.5 User's Guide*, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, Revised 1998.

GAMS/XA

1 Introduction

This document describes the GAMS interface to XA

The GAMS/XA System is the implementation of Sunset Software Technology's XA Callable Library, containing the high performance solvers for LP (linear programming) and MIP (mixed integer programming) problems.

The GAMS/XA Systems offers many options and tuning parameters to solve your models as efficiently as possible. Most of these options are accessible to the GAMS User through an option file, option integer1 through 5, or option real1 through 5. In most cases, GAMS/XA should perform satisfactorily without using any options.

Gamsinst is XA-aware. By default, XA dynamically allocates all available memory, up to 32 megabytes. See GAMS/XA MEMORY USAGE Section to alter this behavior.

2 How to run a model with XA

XA is capable of solving models of the following types: LP, RMIP, and MIP.

If you did not specify XA as a default LP, RMIP or MIP, then you can use the following statement in your GAMS model:

```
OPTION   LP=XA ;   {  RMIP or MIP }
```

It should appear before the SOLVE statement

3 Linear Programming

Linear Programming is a generalized structure for optimally allocating limited resources among competing options. An objective function of decision variables is maximized (or minimized) to determine the allocations to the competing activities via the final values of the decision variables. If fractional values are not acceptable (such as an optimal solution of warehouse to build of 2.3), the problem is a mixed integer problem. The limiting of resources is accomplished by formulating constraint equations that also include decision variables. Linearity must be maintained for the objective function and the constraint equations.

Linear Programs are typically solved through iterative matrix manipulations of the equations (constraints). This identifies successively improving corner points in the feasible region of the problem's solution space. This approach, called the Simplex Method, is also used on mixed integer problems by solving many associated problems to partially enumerate the "decision tree" in an approach called "branch & bound." While this approach can be quite efficient for solving mixed integer problems, its application is limited by the size of the problem it can solve.

XA offers the primal/dual simplex and the barrier algorithm for solving linear problems. The primal/dual simplex method is a very robust method, and in most cases you should get good performance.

XA is very fast, and allows for restarting from an advanced basis. In case you have a GAMS model with multiple solves, and there are relative minor changes between those LP models, the solves proceeding the first one will use basis information from the previous solve to do a "jump start". This is completely automated by GAMS and should be worry-free.

In case of a 'cold start' (the first solve), XA will try to create a better basis than the 'all-slack' basis. The crash routine tries to maintain dual feasibility. Crashing is usually not used for subsequent solves because that would destroy the advanced basis passed from GAMS. The default rule is to crash when GAMS does not pass on a basis, and not to crash otherwise.

The XA presolver is called upon to try to reduce the size of the model. There are some simple reductions that can be applied before the model is solved, like replacing singleton constraints (i.e. X =e= 5;) by the bounds. Although most modelers will already use the GAMS facilities to specify (X.lo and X.up), in many cases there are still possibilities to do these reductions. In addition to GAMS reductions, XA can also remove some redundant rows, and substitute certain constraints. To prevent the 'presolver' from being called enter the following "Set eliminate off" in the xa.opt file.

4 GAMS SEMI-CONTINUOUS VARIABLES

XA supports GAMS' semi-continuous integer and continuous variable types. Semi-continuous variables are variables that are either at zero (0), or greater than or equal to their lower bound, e.g., a pump motor if operating must run at least 2400 r.p.m. and less than 5400 r.p.m., or not at all (0 r.p.m.); or investments levels must exceed a specific threshold or no investment is made.

All semi-continuous variables must have a lower bound specification, e.g., speed.lo(i) = 100. And if the semi-continuous variable has the additional restriction of integer then an upper bound is also required.

Prior to GAMS introduction of this feature, semi-continuous variables had to be emulated by adding one additional binary variable and one additional constraint for each semi-continuous variable. For models of any size, this approach very quickly increase the model's size beyond solvability. Now XA has implicitly defined these variables without the addition of new variables and constraints to your model, hence, increase the size of model that can be solved, in a very neat and clean implementation in our branch and bound algorithm.

For example, to define variables 'a' and 'b' as semi-continuous enter:

```
SemiCont   a ,  b  ;
```

or to define semi-continuous integer variables -

```
SemiInt    y1 , y2 ;
```

GAMS' priority (.prior suffix) values can be associated with semi-continuous variables both integer and continuous types.

All the integer solving options are available and very useful when solving semi-continuous variable because XA's branch and bound algorithms are use to select the direction, either, 0 or greater than it's lower bound value. For example, you can select solving strategies, stop after time, optcr and optca values, etc.

The solve time complexity for semi-continuous variables is comparable with the solve times for binary models and semi-continuous integer variables is comparable to integer solve times.

5 GAMS/XA MEMORY USAGE

The GAMS/XA default memory allocation is 32 Mbytes. If this amount is not available then XA will use whatever is available. This works well on a PC and under Unix, but in certain cases you may want to restrict the amount of memory GAMS/XA uses, for instance:

- If you run Windows and you would like to leave memory for other programs, or

- I you run Windows and you have less than 32 Mbytes of real memory, but a large amount of Virtual memory. In this case XA will request your Virtual memory and initialize it which may be rather time consuming. (You can hear the disk clicking for some time before anything happens).

If you need more memory, follow this procedure

Inside GAMS, add the following line before the solve statement:

```
<modelname>.workspace = xx;
```

where xx is the amount of memory in Mbytes,

In an attempt to insure that all models solve without running out of memory, XA makes one final memory check and if the user supplied memory amount is below what XA would consider reasonable for that size of problem, XA will then increase your amount to XA's minimal value.

Note, that workspace defined in the GAMS model overwrites any memory allocation defined with XAMEMORY. A small XAMEMORY can therefore be used as a default, and it is only overwritten via a GAMS statement for large models.

On multi-processor machines, XA will automatically detect and use all available processors (CPU's) when solving MIP models. You should consider specifying 50% more memory per processor to take full advantage of these processors. One interesting feature of solving parallel MIP models is the potential of superlinear performance.

6 Branch & Bound Strategies

XA is designed to solve a vast majority of LP problems using the default settings. When solving integer problems the default setting may not provide the best for speed and reliability. By experimenting with these parameters performance can be improved dramatically.

6.1 BRANCHING PRIORITIES

Using priorities, when possible, significantly reduce the amount of time required to obtain a good integer solution. If your model has a natural order in time, or in space, or in any other dimension then you should consider using priority branching. For example, multi-period production problem with inventory would use the period value as the priority setting for all variable active in that period, or a layered chip manufacturing process where the priority assigned to binary variable is top down or bottom up in that layer.

If priorities are given to binary, integer, or semi-continuous variables, then these are used to provide a user-specified order for which variables are branched. XA selects the variable with the highest (lowest numerical value) priority for branching and the Strategy determines the direction, up or down.

Priorities are assigned to variables using the .prior suffix. For example:

```
NAVY.PRIOROPT  =  1 ;
...
Z.PRIOR(J,"SMALL")  =  10 ;
Z.PRIOR(J,"MEDIUM")  =  5 ;
Z.PRIOR(J,"LARGE" ) =   1 ;
```

The value 1 indicates the highest priority, and the value 10 the lowest priority. Valid priority values should range between -32000 and 32000. The default priority value is 16000. Columns with the smallest priority value branch and bound first.

6.2 BRANCHING STRATEGIES

Nine 'branch and bound' strategies are provided to meet the demands of many different types of problems. Each strategy has four variations which effects the solve time, speed to first solution, and finding best integer solution. The order in which integer variables are processed during the search for an integer solutions is important. This order is called the 'branching order' of integer variables. Solution times can vary significantly with the method selected.

In general, XA will solve your MIP problems much faster when all model variable are given some kind of objective function value. This biases the basis in a specific direction and usually leads to satisfactory first integer solutions.

The STRATEGY command is available to select the strategy of your choice. The STRATEGY parameter may be specified with one of following values, and assigned to GAMS' INTEGER1 variable, e.g.,

```
OPTION   INTEGER1  =  1  ;
```

Branch & Bound Strategy	Description of Selection Criteria
1	Proprietary method. Default value. Excellent strategy, also add priority to integer variable and try 1P for additional performance gains.
2	Minimum change in the objective function. This strategy has not been very successful at solving MIP problems.
3	Priority based upon column order. This strategy probably does not have must meaning because you cannot set the order variables are generated in GAMS.
4	Column closest to it's integer bound. This strategy tends to send a variable its lower bounds.
6	Column always branches up (high). Second choice after 1. Excellent choice when your model is a multi-period problem; additional performance gains (runs faster) when priority value are equated with period number; also try 6P if using priorities.
7	Column always branches down (low). Useful if variable branched down don't limit capacity or resources. One suggestion is to use priorities in the reverse order described in Strategy 6.
8	Column farthest from it's integer bound. Next to Strategies 1 and 6 this is probably the next choice to try. Using priorities and 8P is also suggested.
9	Column randomly selected, useful when solving very large problems. Priority values helpful is multi-period models.
10	Apparent smoothest sides on the polytope. Priorities helpful.

Each XA Branch and Bound strategy has many variations. Sometimes these variations reduce the solution time but may not yield the optimal integer solution. If you are interested in obtaining a fast and 'good' integer solution (which may not be the optimal integer solution), try these variations.

Variation	Effect
A	This variation reduces the amount of time XA spends estimating the value of a would be integer solution. The values calculated are rough estimates and may eliminate nodes that would lead to better integer solutions. Variation A may not appear with variation B. Example: STRATEGY 1A Or add 100 to your Strategy number and assign to GAMS' INTEGER1 variable, e.g., OPTION INTEGER1 = 101 ; Equates to 1A
B	This variation spends very little time calculating estimated integer solutions at each node and is the most radical in performance and integer solution value and may eliminate nodes that would lead to better integer solutions. Variation B may not appear with variation A. Example: STRATEGY 8B Or add 200 to your Strategy number and assign to GAMS' INTEGER1 variable, e.g., OPTION INTEGER1 = 208 ; Equates to 8B
C	Each time an improving integer solution is found XA splits the remaining node list in half based upon the length of the current list. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation C may not appear with variation D. Example: STRATEGY 6C Or add 400 to your Strategy number and assign to GAMS' INTEGER1 variable, e.g., OPTION INTEGER1 = 406 ; Equates to 6C
D	Each time an improving integer solution is found XA splits the remaining node list based upon the difference in current projected objective and the best possible objective value divided by two. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation D may not appear with variation C. Example: STRATEGY 8D Or add 800 to your Strategy number and assign to GAMS' INTEGER1 variable, e.g., OPTION INTEGER1 = 808 ; Equates to 8D
P	Each time a node is generated XA calculates the effects of each non-integer on future objective function values, which is calculation intensive. By assigning branching priorities to your integer variables XA will only perform this calculation on non-integer variable(s) with the lowest branching priority, which frequently reduces the number of calculations. Example: STRATEGY 1P or: STRATEGY 6BP Variation P may appear any variation, but to be effective you must assign integer branching priorities. Or add 1600 to your Strategy number and assign to GAMS' INTEGER1 variable, e.g., OPTION INTEGER1 = 1606 ; Equates to 6P

7 LIMITSEARCH PARAMETER

LIMITSEARCH is used to limit the number of nodes to search by implicitly or explicitly stating a bound on the value of an integer solution. The integer solution obtained, if any, will have a functional value no worse than LIMITSEARCH. The next integer solution will have a monotonically improving objective function value until an optimal integer solution is found and if verified.

If you can estimate the objective function value of a good integer solution, you can avoid nodes that lead to worse solutions and, consequently, speed up the search. However, too restrictive a value may lead to no integer solution at all, if an integer solution with a functional value better that the LIMITSEARCH value does exist. If the search terminates with 'NO INTEGER SOLUTION', you must begin the search again with a less restrictive LIMITSEARCH value.

The LIMITSEARCH command line parameter has three (3) methods of specifying a lower limit on the objective function.

LIMITSEARCH Value	Meaning
##	Only search for integer solutions between this value and the 'optimal continuous' solution.
##%	Only search for integer solutions with #% of the 'optimal continuous' solution.
(#%)	Solve for the integer solution that is within #% of the 'optimal integer solution'. This can reduce the search time significantly, but the reported integer solution may not be the optimal integer solution but will be within #% of it.

You must experiment with different Strategies and LimitSearch values to determine which is best for your problems. We have found that one of the following settings generally works quite well:

STRATEGY 1 LIMITSEARCH (5%) STOPAFTER 15:00

or

STRATEGY 6 LIMITSEARCH (5%) STOPAFTER 15:00

Also try strategies 1A, 1B, 6A, 6B, and 9. The LimitSearch value should be as large as possible and still meets your business objectives (distance from the optimal integer solution). As you gain experience with these 'strategies', you will be able to make an 'informed' choice.

8 The XA Option File

The option file is called xa.opt. The GAMS model should contain the following line to signal GAMS/XA to use the option file:

```
<modelname>.optfile = 1 ;
```

where <modelname> is the name of the model specified in the model statement. For instance:

```
model  m /all/  ;
m.optfile   =  1 ;
option  LP = XA ;
solve  m  using   LP minimize z ;
```

The XA option file allows you to make solver (XA) specific commands that are not part of the GAMS System. The commands in the XA option file take precedence over their equivalent GAMS command. The option file has the following format: Comment lines beginning with an asterisk (*) . For example:

```
* Integer solving strategy.
   Strategy  6P
* Write log information to the screen every 5 seconds.
   Set   FreqLog  00:05
* Do NOT scale the problem.
     Set  Scale  No
```

The contents of the option file are echoed to the screen.

8.1 Available GAMS/XA Options

Here is a list of the additional GAMS/XA options. Check also the section on the GAMS options in the GAMS User's guide

NAME	Description	Equivalent xa.opt file command
OPTION INTEGER1 = # ;	# is XA's Branch and Bound strategy selection.	Strategy #
OPTION INTEGER2 = # ;	# selects a basis file to use. Values range from 1 to 32000. The default is 0 indicates that no external basis information is available. This basis will override any basis GAMS passes XA. Each GAM's SOLVE statement will cause XA to use and update this basis file. Once a model has been developed (reached it's production phase), use of this capability should improve model re-runs. If XA is solving an integer model and is interrupted by an iteration limit, resource limit, or Control Z, XA will save enough information to restart the branch and bound process just prior to the interruption. In addition, XA automatically saves restarting information every 15 minutes for additional safety against power failure, or accidental human intervention. You can stop and restart a model any number of times and XA will present GAMS with the best solution up to that point. After XA solves a model to its completion, the restarting information is deleted. If you plan to restart your model, do not change it. The size of the restart file is very small, about 50 bytes per integer variable. The restart files extension is r01 and the integer solutions basis file's extension is b01.To restart an interrupted model, just re-run GAMS. GAMS has to reprocess your model statements, but XA will automatically resume where it left off.	Basis filename.ext
OPTION INTEGER3 =	XA terminates after finding # improving integer	Set IntLimit #

NAME	Description	Equivalent xa.opt file command
# ;	solution. Default is 0, indicating no termination. On restarted models the count begins at 0. Normally this termination will leave a restart file.	
OPTION REAL1 = # ;	If an integer solution is found which is better that #, then XA will terminate. The default is 0, indicating no termination value. Normally this termination will leave a restart file.	
OPTION REAL2 = # ;	# indicates the amount of perturbation for highly degenerate problems. A positive value allows XA to generate a uniformly distributive random variable between 0 and #. A negative value uses a constant perturbation of the absolute value of #. The default is 0, indicating no perturbation value. Note: REAL2 options should not be used, except in cases where "all else fails", and definitely not when first starting off. XA has build-in degenerate routines to handle degeneracy.	Set Perturbation #
OPTION REAL3 = # ;	# indicates the number of seconds to continue solving after finding the first integer solution. Normally this termination will leave a restart file. The default is 0, indicating no termination.	StopAfter hh:mm:ss
OPTION REAL4 = # ;	# indicates the maximum percent of binary variables to lock at each generated node. If you believe your model has an integer solution near the relaxed LP solution, then this parameter may help getting a solution faster. We recommend you trying using a value of 5 to 10. This parameter has had mixed success. Default Value: 0, indicates no locking of binary variables.	Set IntPct #

8.2 GAMS/XA Options File Parameters.

8.2.1 Overviev

```
--- Basis / Restarting Information ---
Basis  filename.ext | None | Never          Set ReStart  No | Yes

--- Performance Enhancements ---
Set Pricing  #                              Set Mprice  #
Set Scale  No | Yes | #                      Set Eliminate  No | Yes | Off
Set Perturbate  #                           Set Crash #
Set DegenIter  #                            Set MaxCPU  #

--- Barrier Options ---
Set  Barrier  No | Yes
```

```
--- Reporting Options ---
Output  filename.ext  | CON: |  LPT1:          Set  XARPRT   No  |  Yes
MatList  Variable | Constraint | Both | None   ToMPS  No | Yes

--- Mixed Integer Options ---
Strategy  a                                   LimitSearch   # |  #% | (#%)
Priority  a                                   StopAfter  hh:mm:ss
Set IntGap  #                                 Set IntLimit  #
Set Relaxed  No | Yes                         Set ReStart  No | Yes
Set IntPct  #                                 Set MaxNodes  #
Set Increment  #                              Set Ltolerance  #
Set Utolerance  #
Set BVPriority  #                             Set Iround  No | Yes
TreeDepth  #                                  TreeTime  #
Set Ibounds  #                                Set DualSimplex  No | Yes

--- Stopping Criteria ---
Set Iterations  #                             Set TimeLimit  hh:mm:ss
StopAfter  hh:mm:ss

 --- Algorithmic Tolerances/Enhancements ---
Set Markowitz  #                              Set Pricing  #
Set ReInvertFreq  #                           Set TcTolerance  #
Set Ypivot  #                                 Set XToZero  #
Set ReducedCost  #                            Set ElemSize  #
Set TransEntry  #                             Set RejPivot  #
Set Scale  No | Yes | #

--- System Configuration Options ---
Set Bell  No | Yes
```

8.2.2 Detailed description of options

Parameter	Description
Basis file.ext | none | never	After XA has solved your problem, the solution is saved for the next time the problem is solved. This can greatly reduce the number of iterations and execution time required. The Dual Simplex algorithm is used when XA detects advance basis restarts. You can instruct XA to not use the Dual Simplex algorithm for restarts as follows, Set DualSimplex No. - file.ext - The filename containing an 'advance basis' for restarting. Default file extension is SAV. - none - No 'advance basis' is specified, but the 'final basis' is saved in the problem filename with an extension of SAV. - never - No 'advance basis' is specified, and the final 'basis' is not saved. Defaults to no basis file.
FORCE No | Yes	If your LP model is infeasible, XA makes adjustments in column and row bounds to make the problem feasible. No adjustments are made to binary columns or RHS values of SOS sets. Depending upon how tightly constrained the problem is, XA may be prevented from making additional adjustment that would lead to an integer solution. No adjustments are made to make a column's lower bound less than zero. Default value: NO
LIMITSEARCH # | #% | (#%)	LimitSearch is used to limit the number of nodes to search by implicitly or explicitly stating a upper bound on the integer solution. Default value is no limiting value, or, -1.0e23 for maximizing and 1.0e23 for minimizing.
MATLIST Var | Con | Both | None	Problem is displayed in equation format. This is probably the most useful command when debugging model. - Var - List columns in row. - Con - List rows in columns. - Both - Var and Con - None - no listing (default value)
OUTPUT CON: | filename	Location XA prints all messages. - CON: prints on stdout (default value). - filename with extension prn. Existing files are overridden.
SET Barrier No | Yes | X	Activates XA's primal-dual interior point algorithm. Very use when solving very large scale linear programming models. - Yes - Uses primal-dual interior point algorithm, MIP models automatically crossover the simplex algorithm for branching & bounding. - No – Use the simplex algorithm. - X – Crossover to the simplex algorithm after solving. (automatic when solving MIP models.
SET BELL No | Yes	Turns XA's termination bell on or off. - No - do not ring the bell.(default value). - Yes - ring the bell.
SET CRASH 0 | 1 | 2	Method of generating initial basis. - 0 - Minimize primal infeasibility (default value). - 1 - Minimize dual infeasibility. - 2 - Both 0 & 1.

Parameter	Description
SET DEGENITER #	Degenerate anticycling aide. Number of consecutive degenerate pivots before anti-cycling code is activated. Default value: square root of the number of rows.
Set FREGLOG hh:mm:ss	Frequency in time to print the iteration log line. A negative number (e.g. -00:02) overwrites the same line. This command reduces the overhead of printing too many iteration lines. Default value: 00:01 (one log line per second).
SET INTGAP #	Minimum objective function improvement between each new integer solutions. Reported integer solution may not be the optimal integer solution because of premature termination. Default value 0.00.
SET INTLIMIT #	After finding # improving integer solutions, XA terminates with the best solution thus far. Reported integer solution may not be the optimal integer solution because of premature termination. Defaults - no limit the number of integer solutions.
SET INTPCT #	Percent of available integer columns to consider fixing at each integer node. Useful on very large binary problems. If 100 is entered than all integer columns that are integer at the end of solving the relaxed LP problem are fixed at the current integer bounds. Default is zero (0.0) meaning no fixing.
SET IROUND No | Yes	XA reports either rounded or unrounded integer column primal activity. - YES causes XA to report rounded integer column activity. XA rounds integer activity values to the closest bound based upon the LTOLERANCE and UTOLERANCE values. - NO causes XA to report unrounded integer variable activity, but these activities will always be within the requested integer tolerance. Default value: YES
SET ITERATION #	Maximum number of iteration. XA terminates if limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Reported integer solution may not be the optimal integer solution because of premature termination. Default value: 2000000000
SET LTOLERANCE # SET UTOLERANCE #	The tolerance (closeness) XA uses to solve integer column to it's numeric sequence. For instance, you might consider using an UTOLERANCE of 0.02 (a boat 98% full for all practical purposes to really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then 3 * Y ranges of 3 * 3.95 to 3 * 4.001. Default values: 5.0e-6.
SET MARKOWITZ #	Numeric Stability vs. sparsity in basis updating and inverse. Sparsity favors a larger number at the expensive of numeric stability. Default 10
SET MAXCPU #	Number of processors to use when solving MIP models. In generate, MIP models should solve 1/# times faster that a single processor machine. Consider requesting 50% more memory per processor. Number of processors to utilize. Defaults to the number of processor on the machine. Number can be greater than the number of physical processors.

Parameter	Description
SET MAXNODES #	Maximum number of branch and bound nodes. Default value: 4,000 plus the number of binary variables plus square root of the number of integer columns.
SET PRICING 0 | 1 | 2 | 3 | 4	Variable pricing strategies, useful if XA appears to make a substantial number (rows/2) of pivots that do move the objective function or reduce the sum of infeasibilities. This feature requires more memory because an additional set of matrix coefficient are loaded into memory. - 0 – Standard reduced cost pricing (default value). - 1 – Automatic DEVEX pricing switch over. - 2 – Infeasible DEVEX pricing. - 3 – Feasible DEVEX pricing (our next choice). - 4 - Both infeasible and feasible DEVEX pricing.
SET REDUCEDCOST #	Dual activity tolerance to zero. Default value: 1e-7
SET RELAXED No | Yes	Integer problem is solved as a standard LP and with no integer columns in the formulation. - No – integer problem are solved with branch and bound method (default value). - Yes – solve problems as if all columns where continuous columns.
SET RESTART No | Yes	When solving integer, binary, or semi-continuous problems XA may terminate before exploring all your integer nodes if so you have the option of restarting XA pickup right where you left off. Just before XA returns to your program, XA writes out BASIS command line parameter determines filename used with the appropriate extensions. basis information in the basis file (extension sav). if an integer, binary or semi-continuous solution has been found an integer basis file is created with (extension b01). And if all you integer nodes are not explored then a third file is created for restarting the unchanged problem in (extension r01). - Yes - if an r01 file exists with my identical problem then restart integer problem where I left off and if XA terminates before all nodes are explored again then write restart information to this file. - No - do not use or create the r01 file to restart my problem (default value):
SET SCALE No | Yes | 2	Problem Scaling technique. - No - Data not scaled. - Yes - Column and row scaling (default value). - 2 - Row scaling only.
SET STICKWITHIT #	If an integer basis (.b01) file is reloaded to indicate branching direction for the current XA solve, this branching advice is following until # infeasible branches are made. After # infeasible branches, the standard branching direction for the particular branch & bound strategy is used. Default value 10.
SET TCTOLERANCE #	The smallest technological coefficient allowed in your matrix array. This tolerance is useful when extensive calculations are performed on these coefficients, where the results should be zero (but because of rounding errors) ends up being something like 1.0e-15. Default value: 1.0e-7

Parameter	Description
SET TIMELIMIT hh:mm:ss	Maximum time allow to solving the problem. XA terminates if this limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Units are wall clock time. If set to low, reported integer solution may not be the optimal integer solution because of premature termination. Default value: 2000000000 seconds.
SET XTOZERO #	Primal activity tolerance to zero. Default value: 1e-7
STOPAFTER hh:mm:ss	Amount of time (hh:mm:ss) to continue solving after finding the first integer solution. Reported integer solution may not be the optimal integer solution because of premature termination. Default value: 0, indicating no termination.
STRATEGY # | #A | #B	MIP solving strategy. Default value: 1
TOMPS No | Yes | Secure	Write an MPS file of problem. gams.mps file is created or rewritten. - NO - Do not write an MPS formatted file (default value). - Yes - Write problem in MPS format. - Secure - Write problem in MPS format and change column and row names to: C0, C1,... and R0, R1, ...

9 REPORTS

9.1 STATISTICS REPORT

```
[a] STATISTICS - FILE: alloy  TITLE: Alloy Blending  Mon Jan 01 00:00:00
19xx
[b] XA VERSION xx.x  <---Platform ---> USABLE MEMORY xxxK BYTES
[c] VARIABLES 7  MAXIMUM 100,000
[d]     2 LOWER, 0 FIXED, 5 UPPER, 0 FREE
[e]  CONSTRAINTS 7   MAXIMUM 50,000
[f]     1 GE, 1 EQ, 5 LE, 0 NULL/FREE, 1 RANGED.
[g]  CAPACITY USED BY CATEGORY-
        0.0% VARIABLE,  0.1% CONSTRAINT, 48 NON-ZEROS, WORK 101,977
[h]  MINIMIZATION.

[i] O P T I M A L   S O L U T I O N ---> OBJECTIVE 296.21661
    SOLVE TIME 00:00:00  ITER 12  MEMORY USED   0.1%
```

E x p l a n a t i o n

Starting with the first line in the STATISTICS REPORT:

a) The first line has the problem filename ALLOY, the title for the problem file, today's date and time.
b) Current version of the XA System, platform run on, and the amount of usable memory.
c) Number of variables in the problem.
d) Detailed breakdown of how many variables have upper bounds (=<), fixed (=), lower bounds (>=), free, and integer.
e) Number of constraints in the problem.
f) Detailed breakdown of the number of GE (<=), EQ (=), LE (>=), null/free, and 'ranged' constraints.

g) Capacity used by category - as your problem grows, these percentages will also increase.
h) Statement of the objective of the problem.
i) Status Line. The problem has been solved and these are the results: 1) optimal solution, 2) value of the objective function, 3) iteration time hh:mm:ss, 4) number of iterations, and 5) MEMORY USED -- important to watch as this resource usually grows the fastest. More on 'Status Line' later.

9.2 MATRIX LISTING

The XA System can display your problem in equation format. This is helpful for verifying your formulation and locating errors. This report is controlled with the MATLIST command line parameter --

Example: MatList Variables

```
Minimize
COST: 0.03 BIN1 + 0.08 BIN2 + 0.17 BIN3 + 0.12 BIN4 + 0.15 BIN5 + 0.21 ALUM
   0.38 SILICON

Constraints
YIELD: BIN1 + BIN2 + BIN3 + BIN4 + BIN5 + ALUM + SILICON = 2000
FE: 0.15 BIN1 + 0.04 BIN2 + 0.02 BIN3 + 0.04 BIN4 + 0.02 BIN5 + 0.01 ALUM
   0.03 SILICON <= 60
CU: 0.03 BIN1 + 0.05 BIN2 + 0.08 BIN3 + 0.02 BIN4 + 0.06 BIN5 + 0.01 ALUM
    <= 100
MN: 0.02 BIN1 + 0.04 BIN2 + 0.01 BIN3 + 0.02 BIN4 + 0.02 BIN5 <= 40
MG: 0.02 BIN1 + 0.03 BIN2 + 0.01 BIN5 <= 30
AL: 0.7 BIN1 + 0.75 BIN2 + 0.8 BIN3 + 0.75 BIN4 + 0.8 BIN5 + 0.97 ALUM >=
1500
SI: 0.02 BIN1 + 0.06 BIN2 + 0.08 BIN3 + 0.12 BIN4 + 0.02 BIN5 + 0.01 ALUM
   0.97 SILICON <= 300 >= 250

BIN1 <= 200 | BIN2 <= 750 | 400 <= BIN3 <= 800 | 100 <= BIN4 <= 700 |
BIN5 <= 1500 |
```

Example: MatList Constants

```
Minimize
BIN1: 0.03 COST + YIELD + 0.15 FE + 0.03 CU + 0.02 MN + 0.02 MG + 0.7 AL
   0.02 SI <= 200
BIN2: 0.08 COST + YIELD + 0.04 FE + 0.05 CU + 0.04 MN + 0.03 MG + 0.75 AL
   0.06 SI <= 750
BIN3: 0.17 COST + YIELD + 0.02 FE + 0.08 CU + 0.01 MN + 0.8 AL + 0.08 SI
    >= 400 <= 800
BIN4: 0.12 COST + YIELD + 0.04 FE + 0.02 CU + 0.02 MN + 0.75 AL + 0.12 SI
    >= 100 <= 700
BIN5: 0.15 COST + YIELD + 0.02 FE + 0.06 CU + 0.02 MN + 0.01 MG + 0.8 AL
   0.02 SI <= 1500
ALUM: 0.21 COST + YIELD + 0.01 FE + 0.01 CU + 0.97 AL + 0.01 SI
SILICON: 0.38 COST + YIELD + 0.03 FE + 0.97 SI

YIELD = 2000 | FE <= 60 | CU <= 100 | MN <= 40 | MG <= 30 | AL >= 1500 |
300 <= SI >= 250 |
```

Both layouts may be displayed by entering: MatList Both

9.3 ITERATION LOG LINE

After the Statistical Report is printed, the iteration log line is displayed. The displaying of the iteration log line is controlled by the MUTE command line parameter. MUTE YES or Set FreqLog 0 suppresses displaying of the iteration log line. If the iteration log line is displayed, the following information is shown. The LP iteration log line has three different formats depending upon XA's progress:

```
During LP infeasible iterations -
        Iter: ###   Inf: ####    #
 Iteration count, amount of infeasibility, and number of infeasible
columns.
```

```
During LP feasible iterations -
        Iter: ###  Obj: ###     #
 Iteration count, objective function, and number of columns priced.
```

```
During Integer branch and bound -
Node    IInf    ToGo.Map    Best.Obj    Cur.Obj    Int.Obj    X  Column
U/D
####    ###   #########    #######   #######  ######    #  #######
#
```

	Description
Node	Active node, the smaller the better, value increases and decreases as the branch-and-bound proceeds.
IInf	Number of columns not sequenced aligned. This number converges to 0 as XA approaches an integer solution.
ToGo.Map	A numeric picture of unexplored nodes. Each digits represents the remaining nodes in that digits position. For example, 435 means: • 4 unexplored nodes between nodes 20 and 29. • 3 unexplored nodes between nodes 10 and 19. • 5 unexplored nodes between nodes 0 and 9.
Best.Obj	Best possible integer objective function. As the branch-and-bound algorithm proceeds this number bounds the Optimal Integer Solution. This number does not change very fast.
Cur.Obj	Current objective function. If an integer solution is found it cannot be any better than this value.
Int.Obj	Objective function of the best integer solution found so far. This value improves as additional integer solutions are found.
X	Number of improving integer solutions found.
Column	Column selected by the branch-and-bound process.
U/D	Direction the Column branched, Down(-) or Up(+).

Display of the iteration log line may be toggled on and off by entering a CTRL/U during the iteration process. Use the Set FreqLog command line parameter to minimize the number line displayed. Displaying the iteration log line can significantly slow the solving process down.

9.4 STATUS LINE

After XA solves (or indicates no solution) your problem, you are greeted with one of the following 'status lines':

```
O P T I M A L   S O L U T I O N   - - ->   OBJECTIVE: ####.####
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

```
N O   F E A S I B L E   S O L U T I O N
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

Identifies constraints and amount(s) of adjustments to make the problem 'feasible'. If FORCE YES is set, then XA automatically makes these adjustments and continues.

```
U N B O U N D E D   S O L U T I O N
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

The unbounded variable is identified. If FORCE YES is set, then XA automatically sets variable to lower limit and continues.

```
U S E R   L I M I T S   E X C E E D E D
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

Entering a CTRL/Z terminates XA and the current solution is saved for restarting.

```
T I M E   L I M I T   E X C E E D E D
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

The time allowed to solve this problem was exceeded. The current solution is saved for restarting.

```
W O R K S P A C E   S I Z E   E X C E E D E D
SOLVE TIME hh:mm:ss  ITER xx  MEMORY USED   x.x%
```

Problem too large. See GAM/XA MEMORY USAGE Section.

If the problem terminates with anything other than 'optimal solution' then the 'title' is changed to reflect the reason.

10 SOLUTION INTERRUPTION

This feature gives you complete control over the operation of your computer. Long running problems may be interrupted without losing the time already spent on solving the problem. This feature can be used as a checkpoint for restarting in event of a power failure, etc. d A problem may be interrupted any number of times, plus you may change your original problem any way you wish. The XA System will make the appropriate changes and solve the newly modified problem. This feature may NOT be used if you enter BASIS NEVER as an XA command line parameter

To interrupt your problem, simply type a CTRL/Z (a Z on Unix platforms) during the 'iteration' process. The XA System will save the current solution into filename.sav (because the current iteration must complete its run, there may be a slight delay before XA responds to your CTRL/Z). When you interrupt XA with a CTRL/Z the following message is displays.

```
U S E R   L I M I T S   E X C E E D E D
```

To restart your problem, simply re-enter what you originally entered to execute GAMS.

GAMS/XPRESS

Contents

1 Introduction

This document describes the GAMS/XPRESS linear and mixed-integer programming solver. The GAMS/XPRESS solver is based on the XPRESS-MP Optimization Subroutine Library, and runs only in conjunction with the GAMS modeling system.

GAMS/XPRESS (also simply referred to as XPRESS) is a versatile, high - performance optimization system. The system integrates a powerful simplex-based LP solver, a MIP module with cut generation for integer programming problems and a barrier module implementing a state-of-the-art interior point algorithm for very large LP problems.

The GAMS/XPRESS solver is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes XPRESS, there is no size or algorithm restriction imposed by the license, nor is any separate licensing procedure required.

2 Usage

If you have installed the system and configured XPRESS as the default LP, RMIP[1] and MIP solver, all LP, RMIP and MIP models without a specific solver option will use XPRESS. If you installed another solver as the default, you can explicitly request a particular model to be solved by XPRESS by inserting the statement

```
option LP = xpress; { or MIP or RMIP }
```

somewhere before the `solve` statement.

The following standard GAMS options can be used to control XPRESS:

- `option reslim =x;` or `modelname.reslim = x;`

  Stop the algorithm after `x` seconds and report the best solution found so far. `Modelname` is the name of the model as specified in a previous `model` statement.

[1] RMIP means: Relaxed Mixed Integer Programming. You can solve a MIP model as an RMIP. This will ignore the integer restrictions and thus solves the problem as an LP.

- `option iterlim=n;` or `modelname.iterlim = n;`

  Stop the algorithm after n simplex iterations and report the best solution found so far. For MIP models, this places a cumulative limit on simplex iterations for the relaxed LP and the nodes of the B&B tree. `Modelname` is the name of the model as specified in a previous `model` statement.

- `option sysout=on;`

  Echo more detailed information about the solution process to the listing file.

- `option optcr=x;`

  In a MIP stop the search as soon as the relative gap is less than `x`.

- `option optca=x;`

  In a MIP stop the search as soon as the absolute gap is less than `x`.

- `option bratio=x;`

  Determines whether or not an advanced basis is passed on to the solver. `Bratio=1` will always ignore an existing basis (in this case XPRESS will use a crash routine to find a better basis than an all-slack basis), while `bratio=0` will always accept an existing basis. Values between 0 and 1 use the number of non-basic variables found to determine if a basis is likely to be good enough to start from.

- `modelname.prioropt = 1;`

  Turns on usage of user-specified priorities. Priorities can be assigned to integer and binary variables using the syntax: `variablename.prior = x;`. Default priorities are 0.0. Variables with a priority v1 are branched upon earlier than variables with a priority `v2` if `v1<v2`.

- `modelname.nodlim = n;`

  Specifies a node limit for the Branch-and-Bound search. When the number of nodes exceeds this number the search is stopped and the best integer solution found (if any) is reported back to GAMS. The default value of 0 indicates: no node limit.

In general this is enough knowledge to solve your models. In some cases you may want to use some of the XPRESS options to gain further performance improvements or for other reasons.

3 Options

Options can be specified in a file called *xpress.opt*. The syntax is rather simple: a line in the option file can be one of the following:

- An empty line or a line consisting only of blanks.
- A comment line, which is a line in which the first non-blank character is an asterisk '*'. The remainder of the line is ignored.
- An option, which consists of a keyword followed by a value.

An example of a valid option file is:

```
* sample XPRESS options file
algorithm simplex
presolve   0
IterLim    50000
```

Keywords are not case sensitive. I.e. whether you specify `iterlim`, `ITERLIM`, or `Iterlim` the same option is set. To use an options file you specify a model suffix `modelname.optfile=1;` or use command line options `optfile=1`.

The tables that follow contain the XPRESS options. They are organized by function (e.g. LP or MIP) and also by type: some options control the behavior of the GAMS/XPRESS link and will be new even to experienced XPRESS users, while other options exist merely to set control variables in the XPRESS library and may be familiar to XPRESS users.

3.1 General Options

The following general options control the behavior of the GAMS/XPRESS link.

Option	Description	Default
`advbasis`	`0:` don't use advanced basis provided by GAMS `1:` use advanced basis provided by GAMS This option overrides the GAMS BRATIO option.	Determined by GAMS
`algorithm`	`simplex:` use simplex solver `barrier:` use barrier algorithm This option is used to select the barrier method to solve LP's. By default the barrier method will do a crossover to find a basic solution.	`simplex`
`basisout`	If specified an MPS basis file is written. In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.	Don't write a basis file.
`iterlim`	Sets the iteration limit for simplex algorithms. When this limit is reached the system will stop and report the best solution found so far. Overrides the GAMS ITERLIM option.	`10000`
`mpsoutputfile`	If specified XPRESS-MP will generate an MPS file corresponding to the GAMS model. The argument is the file name to be used. It can not have an extension: XPRESS-MP forces the extension to be .MAT even if an extension was specified. You can prefix the file name with a path.	Don't write an MPS file.
`rerun`	Applies only in cases where presolve is turned on and the model is diagnosed as infeasible or unbounded. If rerun is nonzero, we rerun the model using primal simplex with presolve turned off in hopes of getting better diagnostic information. If rerun is zero, no good diagnostic information exists, so we return no solution, only an indication of unboundedness/infeasibility.	`1`
`reslim`	Sets the resource limit. When the solver has used more than this amount of CPU time (in seconds) the system will stop the search and report the best solution found so far. Overrides the GAMS RESLIM option.	`1000.0`

The following general options set XPRESS library control variables, and can be used to fine-tune XPRESS.

Option	Description	Default
`crash`	A crash procedure is used to quickly find a good basis. This option is only relevant when no advanced basis is available. `0:` no crash `1:` singletons only (one pass) `2:` singletons only (multi-pass) `3:` multiple passes through the matrix considering slacks `4:` multiple passes (<= 10), but do slacks at the end `>10:` as 4 but perform n-10 passes `100:` default crash behavior of XPRESS-MP version 6	`0` when GAMS provides an advanced basis, and `2` otherwise
`extrapresolve`	The initial number of extra elements to allow for in the presolve. The space required to store extra presolve elements is allocated dynamically, so it is not necessary to set this control. In some cases, the presolve may terminate early if this is not increased.	`automatic`

Option	Description	Default
lpiterlimit	Sets the iteration limit for simplex algorithms. For MIP models, this is a per-node iteration limit for the B&B tree. Overrides the iterlim option.	MAXINT
mpsnamelength	Maximum length of MPS names in characters. Internally it is rounded up to the smallest multiple of 8. MPS names are right padded with blanks. Maximum value is 64.	0
presolve	-1: Presolve applied, but a problem will not be declared infeasible if primal infeasibilities are detected. The problem will be solved by the LP optimization algorithm, returning an infeasible solution, which can sometimes be helpful. 0: Presolve not applied. 1: Presolve applied. 2: Presolve applied, but redundant bounds are not removed. This can sometimes increase the efficiency of the barrier algorithm. As XPRESS does a basis preserving presolve, you don't have to turn off the presolver when using an advanced basis.	1
scaling	Bitmap to determine how internal scaling is done. If set to 0, no scaling will take place. The default of 35 implies row and column scaling done by the maximum element method. Bit 0 = 1: Row scaling. Bit 1 = 2: Column scaling. Bit 2 = 4: Row scaling again. Bit 3 = 8: Maximin. Bit 4 = 16: Curtis-Reid. Bit 5 = 32: Off implies scale by geometric mean, on implies scale by maximum element. Not applicable for maximin and Curtis-Reid scaling.	35
trace	Control of the infeasibility diagnosis during presolve - if nonzero, the infeasibility will be explained. This output appears on the log file, and is best understood if you also set mpsnamelength to a positive value.	0

3.2 LP Options

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS LP solver.

Option	Description	Default
bigmmethod	0: for phase I / phase II 1: if 'big M' method to be used	automatic
bigm	The infeasibility penalty used in the "Big M" method	automatic
defaultalg	1: automatic 2: dual simplex 3: primal simplex 4: Newton barrier	1
etatol	Zero tolerance on eta elements. During each iteration, the basis inverse is premultiplied by an elementary matrix, which is the identity except for one column: the eta vector. Elements of eta vectors whose absolute value is smaller than etatol are taken to be zero in this step.	1.0e-12
feastol	This is the zero tolerance on right hand side values, bounds and range values. If one of these is less than or equal to feastol in absolute value, it is treated as zero.	1.0e-6
invertfreq	The frequency with which the basis will be inverted. A value of -1 implies automatic.	-1
invertmin	The minimum number of iterations between full inversions of the basis matrix.	3

Option	Description	Default
lplog	The frequency at which the simplex iteration log is printed. n < 0: detailed output every -n iterations n = 0: log displayed at the end of the solution process n > 0: summary output every n iterations	100
matrixtol	The zero tolerance on matrix elements. If the value of a matrix element is less than or equal to matrixtol in absolute value, it is treated as zero.	1.0e-9
optimalitytol	This is the zero tolerance for reduced costs. On each iteration, the simplex method searches for a variable to enter the basis which has a negative reduced cost. The candidates are only those variables which have reduced costs less than the negative value of optimalitytol.	1.0e-6
penalty	Minimum absolute penalty variable coefficient used in the "Big M" method.	automatic
pivottol	The zero tolerance for matrix elements. On each iteration, the simplex method seeks a nonzero matrix element to pivot on. Any element with absolute value less than pivottol is treated as zero for this purpose.	1.0e-9
pricingalg	This determines the pricing method to use on each iteration, selecting which variable enters the basis. In general Devex pricing requires more time on each iteration, but may reduce the total number of iterations, whereas partial pricing saves time on each iteration, although possibly results in more iterations. -1:if partial pricing is to be used 0: if the pricing is to be decided automatically. 1: if DEVEX pricing is to be used	0
relpivottol	At each iteration a pivot element is chosen within a given column of the matrix. The relative pivot tolerance, relpivottol, is the size of the element chosen relative to the largest possible pivot element in the same column.	1.0e-6

3.3 MIP Options

In some cases, the branch-and-bound MIP algorithm will stop with a proven optimal solution or when unboundedness or (integer) infeasibility is detected. In most cases, however, the global search is stopped through one of the generic GAMS options:

1. iterlim (on the cumulative pivot count), reslim (in seconds of CPU time),
2. optca & optcr (stopping criteria based on gap between best integer solution found and best possible) or
3. nodlim (on the total number of nodes allowed in the B&B tree).

It is also possible to set the maxnode and maxmipsol options to stop the global search: see the table of XPRESS control variables for MIP below.

The following options control the behavior of the GAMS/XPRESS link on MIP models.

Option	Description	Default
mipcleanup	If nonzero, clean up the integer solution obtained, i.e. round and fix the discrete variables and re-solve as an LP to get some marginal values for the discrete vars.	1
miptrace	A miptrace file with the specified name will be created. This file records the best integer and best bound values every miptracenode nodes and at miptracetime-second intervals.	none
miptracenode	Specifies the node interval between entries to the miptrace file.	100
miptracetime	Specifies the time interval, in seconds, between entries to the miptrace file.	1

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.

Option	Description	Default
backtrack	This determines how the next node in the tree search is selected for processing. 1: If miptarget is not set, choose the node with the best estimate. If miptarget is set (by the user or by the global search previously finding an integer solution), the choice is based on the Forrest-Hirst-Tomlin Criterion, which takes into account the best current integer solution and seeks a new node which represents a large potential improvement. 2: Always choose the node with the best estimated solution. 3: Always choose the node with the best bound on the solution.	3
breadthfirst	If nodeselection = 4, this determines the number of nodes to include in a breadth-first search.	10
covercuts	The number of rounds of lifted cover inequalities at the top node. A lifted cover inequality is an additional constraint that can be particularly effective at reducing the size of the feasible region without removing potential integral solutions. The process of generating these can be carried out a number of times, further reducing the feasible region, albeit incurring a time penalty. There is usually a good payoff from generating these at the top node, since these inequalities then apply to every subsequent node in the tree search.	20
cpmaxcuts	The initial maximum number of cuts that will be stored in the cut pool. During optimization, the cut pool is subsequently resized automatically.	100
cpmaxelems	The initial maximum number of nonzero coefficients which will be held in the cut pool. During optimization, the cut pool is subsequently resized automatically.	200
cutdepth	Sets the maximum depth in the tree search at which cuts will be generated. Generating cuts can take a lot of time, and is often less important at deeper levels of the tree since tighter bounds on the variables have already reduced the feasible region. A value of 0 signifies that no cuts will be generated.	0
cutfreq	This specifies the frequency at which cuts are generated in the tree search. If the depth of the node modulo cutfreq is zero, then cuts will be generated.	8
cutstrategy	This specifies the cut strategy. An aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored. -1: Automatic selection of either the conservative or aggressive cut strategy. 0: No cuts. 1: Conservative cut strategy. 2: Aggressive cut strategy.	-1
degradefactor	Factor to multiply estimated degradations associated with an unexplored node in the tree. The estimated degradation is the amount by which the objective function is expected to worsen in an integer solution that may be obtained through exploring a given node.	1.0
gomcuts	The number of rounds of Gomory cuts at the top node. These can always be generated if the current node does not yield an integral solution. However, Gomory cuts are not usually as effective as lifted cover inequalities in reducing the size of the feasible region.	2
maxmipsol	This specifies a limit on the number of integer solutions to be found (the total number, not necessarily the number of solutions with distinct objectives). 0 means no limit.	0
maxnode	The maximum number of nodes that will be explored. If the GAMS nodlim model suffix is set, that setting takes precedence.	100,000,000

Option	Description	Default
mipabscutoff	If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to mipabscutoff. This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.	automatic
mipaddcutoff	The amount to add to the objective function of the best integer solution found to give the new cutoff. Once an integer solution has been found whose objective function is equal to or better than mipabscutoff, improvements on this value may not be interesting unless they are better by at least a certain amount. If mipaddcutoff is nonzero, it will be added to mipabscutoff each time an integer solution is found which is better than this new value. This cuts off sections of the tree whose solutions would not represent substantial improvements in the objective function, saving processor time. Note that this should usually be set to a negative number for minimization problems, and positive for maximization problems. Notice further that the maximum of the absolute and relative cut is actually used.	0.0
miplog	Global print control 0: No printout in global. 1: Only print out summary statement at the end. 2: Print out detailed log at all solutions found. 3: Print out detailed eight-column log at each node. n < 0: Print out summary six-column log at each -n nodes, or when a new solution is found	-100
mippresolve	Bitmap determining type of integer processing to be performed. If set to 0, no processing will be performed. Bit 0: Reduced cost fixing will be performed at each node. This can simplify the node before it is solved, by deducing that certain variables' values can be fixed based on additional bounds imposed on other variables at this node. Bit 1: Logical preprocessing will be performed at each node. This is performed on binary variables, often resulting in fixing their values based on the constraints. This greatly simplifies the problem and may even determine optimality or infeasibility of the node before the simplex method commences. Bit 2: Probing of binary variables is performed at the top node. This sets certain binary variables and then deduces effects on other binary variables occurring in the same constraints.	automatic
miprelcutoff	Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of mipabscutoff. The effect is to cut off the search in parts of the tree whose best possible objective function would not be substantially better than the current solution.	0.0
miptarget	The target objective value for integer solutions. This is automatically set after solving the LP unless set by the user.	1e40
miptol	This is the tolerance within which a decision variable's value is considered to be integral.	5.0e-6
nodeselection	This determines which nodes will be considered for solution once the current node has been solved. 1: *Local first*: Choose among the two descendant nodes, if none among all active nodes. 2: *Best first*: All nodes are always considered. 3: *Depth first*: Choose deepest node, but explore two descendents first. 4: *Best first, then local first*: All nodes are considered for the first breadthfirst nodes, after which the usual default behavior is resumed.	automatic

Option	Description	Default
pseudocost	The default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search. A pseudo cost is associated with each integer decision variable and is an estimate of the amount by which the objective function will be worse if that variable is forced to an integral value.	0.01
treecovercuts	The number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree. Compare with the description for covercuts.	2
treegomcuts	The number of rounds of Gomory cuts generated at nodes other than the first node in the tree. Compare with the description for gomcuts.	0
varselection	This determines how to combine the pseudo costs associated with the integer variables to obtain an overall estimated degradation in the objective function that may be expected in any integer solution from exploring a given node. It is relevant only if backtrack has been set to 1. 1: Sum the minimum of the 'up' and 'down' pseudo costs. 2: Sum all of the 'up' and 'down' pseudo costs. 3: Sum the maximum, plus twice the minimum of the 'up' and 'down' pseudo costs. 4: Sum the maximum of the 'up' and 'down' pseudo costs. 5: Sum the 'down' pseudo costs. 6: Sum the 'up' pseudo costs.	1

3.4 Newton-Barrier Options

The barrier method is invoked by using one of the options

```
algorithm       barrier
defaultalg      4
```

The barrier method is likely to use more memory than the simplex method. No warm start is done, so if an advanced basis exists, you may not wish to use the barrier solver.

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS barrier solver.

Option	Description	Default
bariterlimit	Maximum number of barrier iterations	200
crossover	Determines whether the barrier method will cross over to the simplex method when at optimal solution has been found, in order to provide an end basis. 0: No crossover. 1: Crossover to a basic solution.	1

4 Helpful Hints

The comments below should help both novice and experienced GAMS users to better understand and make use of GAMS/XPRESS.

- **Infeasible and unbounded models** The fact that a model is infeasible/unbounded can be detected at two stages: during the presolve and during the simplex or barrier algorithm. In the first case we cannot recover a solution, nor is any information regarding the infeasible/unbounded constraint or variable provided (at least in a way that can be returned to GAMS). In such a situation, the GAMS link will automatically rerun the model using primal simplex with presolve turned off (this can be avoided by setting the rerun option to 0). It is possible (but very unlikely) that the simplex method will solve a model to optimality while the presolve claims the model is infeasible/unbounded (due to feasibility tolerances in the simplex and barrier algorithms).

- The barrier method does not make use of `iterlim`. Use `bariterlim` in an options file instead. The number of barrier iterations is echoed to the log and listing file. If the barrier iteration limit is reached during the barrier algorithm, XPRESS continues with a simplex algorithm, which will obey the `iterlim` setting.
- Semi-integer variables are not implemented in the link, nor are they supported by XPRESS; if present, they trigger an error message.