CONOPT

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

Contents

1 Introduction

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there is a large number of different solvers available and the number is growing.

The most important distinction between the solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution (so called Global Solvers) can usually not solve very large models. As a contrast most Local Solvers can work with much larger models, and models with over 10,000 variables and constraints are not unusual. If the model has the right mathematical properties, e.g. is convex, then Local Solvers will find a global optimum. Unfortunately, the mathematical machinery for testing whether a general NLP model is convex or not has not yet been developed (and is expected to be in the class or hard problems).

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. When GAMS is installed you must select one of the nonlinear programming algorithms as the default solver for NLP models. If you want to switch between algorithms for a particular model you may add the statement "OPTION NLP = <solvername>;", in your GAMS source file before the SOLVE statement, you may add NLP = <solvername>, on the GAMS command line, or you may change the defaults using the GAMS IDE or by rerunning the GAMSINST 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/CONOPT is well suited for models with very nonlinear constraints. If you experience that a solver has problems maintaining feasibility during the optimization you should try CONOPT. On the other hand, if you have a model with few nonlinearities outside the objective function then other solvers could be the best solver.

GAMS/CONOPT 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 CONOPT. CONOPT can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System.

GAMS/CONOPT can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT. IPOPT and KNITRO will also use second derivatives, but the method is very different and it is not possible to predict which solver will be better.

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

GAMS/CONOPT 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 100,000 equations and variables have been solved successfully, and CNS models with over 1,000,000 equations and variables have also been solved. The components used to build CONOPT have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT can also be used for denser models, but the performance will suffer significantly.

GAMS/CONOPT 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, CONOPT will use the same algorithm used for a real NLP model and it will search for a point that satisfies standard first-order optimality conditions without taking into account that parts of the model could be non-smooth or non-differentiable. The lack of smoothness may confuse the algorithm in CONOPT causing slow convergence, and a point that satisfies standard first-order optimality conditions may not even exist. There are therefore no guaranties whatsoever for this class of models. If CONOPT terminates with a locally optimal solution then the solution will indeed be locally optimal. However, you will sometimes 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. The final point may or may not be locally optimal. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section 7.

Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT 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 CONOPT algorithm has therefore been moved to an appendix (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 CONOPT 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 CONOPT 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).

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:

```
CONOPT 3         Jul  4, 2012 23.9.4 WEX 35892.35906 WEI x86_64/MS Windows


   C O N O P T 3   version 3.15G
   Copyright (C)   ARKI Consulting and Development A/S
                   Bagsvaerdvej 246 A
                   DK-2880 Bagsvaerd, Denmark


  Iter Phase Ninf   Infeasibility   RGmax     NSB   Step InItr MX OK
     0    0          1.6354151782E+01 (Input point)
                             Pre-triangular equations:       2
                             Post-triangular equations:      1
     1    0          1.5354151782E+01 (After pre-processing)
     2    0          3.0983571843E+00 (After scaling)
    10    0     12   3.0814290456E+00                0.0E+00    T  T
    20    0     12   3.0814290456E+00                0.0E+00    T  T
    30    0     13   3.0814290456E+00                0.0E+00    F  F
    40    0     18   2.3738740159E+00                2.3E-02    T  T
    50    0     23   2.1776589484E+00                0.0E+00    F  F


  Iter Phase Ninf   Infeasibility   RGmax     NSB   Step InItr MX OK
    60    0     33   2.1776589484E+00                0.0E+00    T  T
    70    0     43   2.1776589484E+00                0.0E+00    F  F
    80    0     53   2.1776589484E+00                0.0E+00    F  F
    90    0     63   2.1776589484E+00                0.0E+00    F  F
   100    0     73   2.1776589484E+00                0.0E+00    F  F
   110    0     83   2.1776589484E+00                0.0E+00    F  F
   120    0     93   2.1776589484E+00                0.0E+00    F  F
   130    0    103   2.1776589484E+00                0.0E+00    F  F
   140    0    113   2.1776589484E+00                0.0E+00    T  T
   150    0    119   8.7534351971E-01                0.0E+00    F  F


  Iter Phase Ninf   Infeasibility   RGmax     NSB   Step InItr MX OK
   160    0    124   9.5022881759E-01                0.0E+00    F  F
```

```
    170   0   134   9.5022881759E-01                0.0E+00        F  F
    180   0   144   9.5022881759E-01                0.0E+00        F  F
    190   0   154   9.5022881759E-01                0.0E+00        F  F
    201   1   160   9.4182618946E-01 4.3E+01    134 2.4E-06        T  T
    206   1   130   8.2388503304E-01 9.5E+01    138 1.0E+00     13 T  T
    211   1    50   1.0242911941E-01 6.9E+00     84 7.2E-01     24 T  T
    216   1    16   2.6057507770E-02 1.3E+00     52 6.1E-01     17 T  T
    221   1     5   7.2858773666E-04 6.1E-03     38 6.0E-01      7 F  F

 ** Feasible solution. Value of objective =     1.00525015566

   Iter Phase Ninf      Objective     RGmax    NSB    Step InItr MX OK
    226   3          1.0092586645E+00 4.4E-04    38 1.0E+00      3 T  T
    231   3          1.0121749760E+00 1.4E+00    24 4.8E-01      9 T  T
    236   3          1.0128148550E+00 4.8E-06    13 5.8E-02     12 F  T
    241   3          1.0128161551E+00 2.5E-06    12 9.1E+03        F  T
    246   4          1.0128171043E+00 1.2E-07    13 1.0E+00      3 F  T
    247   4          1.0128171043E+00 5.7E-08    13

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

The first few lines identify the version of CONOPT you use.

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

The remaining iterations are characterized by the value of "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 (labeled "Infeasibility") is being minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4 (now labeled "Objective"), is minimized or maximized. Phase 0 iterations are Newton-like iterations. They are very cheap so you should not be concerned if there are many of these Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by a number of inner SLP iterations in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinear and most aspects of the iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. For small and simple models CONOPT will approximate second order information as a byproduct of the line searches. For larger and more complex models CONOPT will use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These SQP iterations are identified by the number of inner SQP iterations in the "InItr" column.

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 reduced gradient among the non-optimal variables. Rgmax should eventually converge towards zero, but convergence is not expected to be monotone. The last two columns labeled "MX" and "OK" gives information about the line search: OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated before an optimal step length was found because it was not possible to find a feasible solution for large step lengths. MX = T means that the line search was terminated by a variable reaching a bound (always combined with OK = T), and MX = F means that the step length was determined by nonlinearities. If OK = T then the step length was optimal; if OK = F then the constraints were too nonlinear to allow CONOPT to make a full optimal step.

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 1.e-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. For this to happen 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 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 6.4 for hints on model scaling. Finally, it can be caused by stalling as described in section A13.4 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 of `rtmaxv` (default 1.e10) and it returns with Modelstat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. 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 have to make a line search and move to a region with large values of the variables. This may lead to bad scaling and to many different kinds of tolerance and roundoff problems, including problems of determining whether a solution is feasible or not.

The message above exists 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 1.02E+10 and current upper bound
      (Rtmaxv) is 1.00E+10
```

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 2000000000 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 even after a very small change 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 6.4. 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>.WORKFACTOR = x.x;" or "<model>.WORKSPACE = xx;" in GAMS or by adding "workfactor=xx" to the command line call of 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%
            1st derivative evaluations    0.000 =  0.0%
```

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 = 13 (System 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 11 (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.

It is recommended that you use the WORKFACTOR option if you must change the amount of memory. The same number will usually work for a whole family of models. If you prefer to use `WORKSPACE`, the GAMS `WORKSPACE` option corresponds to the amount of memory, measured in Mbytes.

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 variable bounds. 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 the system to crash. 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 "conopt.opt". 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 or with `optfile = 1` on the command line.

The format of the CONOPT Options file consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e12
lfnsup = 500
```

Upper case letters are converted to lower case so the second line could also be written as "LFNSUP = 500". The value must be written using legal GAMS format, i.e. a real number may contain an optional E exponent, but a number may not contain blanks. The value must have the same type as the option, i.e. real options must be assigned real values, integer options must be assigned integer values, and logical options must be assigned logical values. The logical value representing true are `true, t, yes`, or `1`, and the logical values representing false are `false, f, no`, or `0`.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported. You can still replace the equal sign with `:=` and you can add end of line comments after a `#` or `!` character. Lines starting with `*` in column 1 are treated as comment lines.

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.L(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. You may also try the optional Crash procedure described in section A4.3 in Appendix A by adding the line "lstcrs=t" to the CONOPT 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 CONOPT 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. If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

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 three reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. Second, because the lower bound on the intermediate result will bound the search away from the singularity at XSUM = 0. And third, because the matrix of second derivatives for the last model only depend on XSUM while it depends on all X in the first model.

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 invertible increasing function you may even rewrite the last constraint as a simple bound:

```
CAP.LO(I) = finv(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   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.

6.4.1 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.

6.4.2 Using the Scale Option in 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 model, 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 are a good approximations 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 depend 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 $\geq$ 0) and (fplus $\geq$ 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.e-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.

8   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 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). If the model is "very" nonlinear and has many degrees of freedom an improved search direction (step 7) can be computed using specialized inner SQP-like iterations based on exact second derivatives for the model (section A11).

The remaining two sections give some short guidelines for selecting non-default options (section A12), and discuss miscellaneous topics (section A13) such as CONOPT's facilities for strictly triangular models (A13.1) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear Systems (A13.2), as well as numerical difficulties due to loss of feasibility (A13.3) and slow or no progress due to stalling (A13.4).

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 intermediate 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 display 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. Constraint-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, equations and variables are ordered as shown in Fig. 4.1.

A4.1   Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Fig. 4.1. 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.
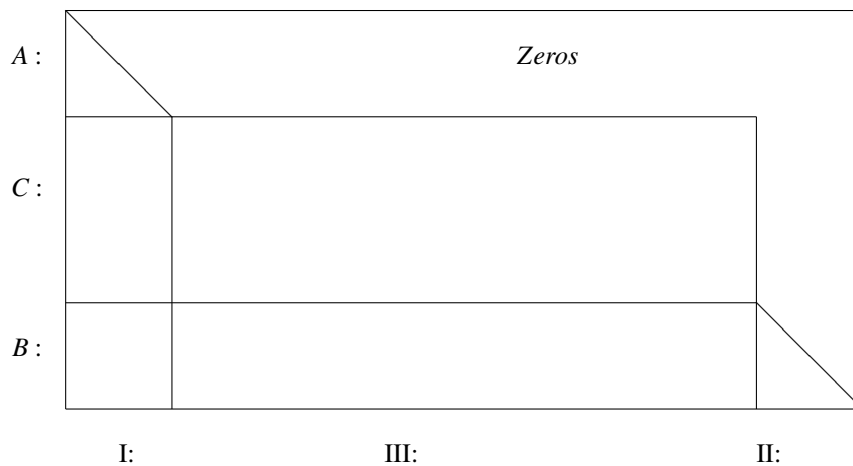
Figure 4.1: The ordered Jacobian after Preprocessing.

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 A13.1.

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 nonlinear 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. 4.1. 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 CONOPT 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 computation of both first and second derivatives 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.

It is not necessary to have a coefficient of 1 for the variable to be substituted out in the post-triangular phase. However, a non-zero coefficient cannot be smaller than the absolute pivot tolerance used by CONOPT, `Rtpiva`.

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
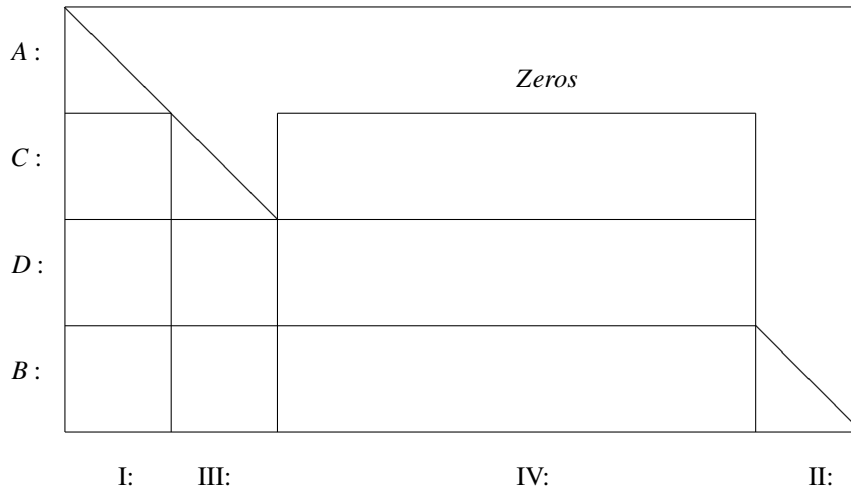
Figure 4.2: The ordered Jacobian after Preprocessing and Crashing.

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.

Fig. 4.2 shows a reordered version of Fig. 4.1. 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. 4.2. There is, as already shown with the small example in section A4.2 above, no guarantee 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.

A5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default is to turn scaling 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 given mixed results with some advantage for scaling, and scaling is therefore by default turned on, corresponding to the CONOPT option "lsscal = t". 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 6.4 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. 4.1) by scale factors computed as follows:

1. CONOPT computes the largest term for each constraint, i. This is defined as the maximum of the constant right hand side, the slack (if any), and abs(Jac(i,j)*X(j)) where Jac(i,j) is the derivative and X(j) is the variable.

2. The constraint scale factor is defined as the largest term in the constraint, projected on the interval [Rtmins, Rtmaxs]. The constraint is then divided by the constraint scale factor. Ignoring the projection, the result is a model in which the largest term in each constraint is exactly 1. The purpose of the projection is to prevent extreme scaling. The default value of Rtmins is 1 which implies that we do not scale the constraints up. Constraints with only small terms remain unchanged. The default value of Rtmaxs is 2**30 or around 1.07e9 so terms much larger than 1.e9 are only partially scaled down and will still remain large.

3. The terms abs(Jac(i,j)*X(j)) after constraint scaling measure the importance of each variable in the particular constraint. The variable scale is selected so the largest importance of the variable over all constraints is 1. This gives a very simple variable scale factor, namely the absolute value of the variable. The variable is then divided by the variable scale factor. To avoid extreme scaling we again project on the interval [Rtmins, Rtmaxs]. Variables less than Rtmins (default 1) are therefore not scaled up and variables over Rtmaxs (default 2**30 = 1.07e9) are only partially scaled down.

All scale factors are rounded down to a power of 2 to preserve precision in the internal computations. To avoid difficulties with rapidly varying variables and derivatives CONOPT recomputes the scale factors at regular intervals (see lfscal).

The options that control scaling, lsscal, lfscal, rtmins, and rtmaxs, are all described in Appendix B.

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 options `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 phases 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phases 3 and 4, which are 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 referred 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 "InItr" in the iteration log. The number in the InItr 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 *conopt.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 InItr column.

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 *conopt.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.

A11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities CONOPT needs second order information. Some second order information can be derived from the line search and is used in the first iterations in Phase 2 or 4. Depending on progress, CONOPT may switch to a Sequential Quadratic Programming (SQP) procedure that works on a sub-model with linear constraints and a quadratic objective function. The constraints are a linearization of the nonlinear constraints, and the objective function is derived from the Hessian of the Lagrangian function. CONOPT will inside the SQP procedure use exact second order information computed by GAMS. The result of the SQP procedure is a search direction and a basis and CONOPT will afterwards use the same line search procedure and feasibility preserving steps as after the SLP procedure. CONOPT remains a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SQP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The number in the InItr column is the number of non-degenerate SQP iterations. The effort spend inside the SQP procedure is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

The SQP 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 of the constraints. This is indicated with MX = T and OK = T.

- The objective is much more nonlinear along the search direction than expected and the optimal step is not one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SQP procedure or not, based on progress information. You may turn it off completely with the line "lsesqp = f" in the CONOPT options file (usually conopt.opt). The default value of lsesqp (lsesqp = Logical Switch Enabling SQP mode) is t or true, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define lsesqp, but it can be used for experimentation.

In connection with 1st and 2nd derivatives the listing file (*.lst) will have a few extra lines. The first looks as follows:

```
The model has 537 variables and 457 constraints
```

```
    with 1597 Jacobian elements, 380 of which are nonlinear.
    The Hessian of the Lagrangian has 152 elements on the diagonal,
    228 elements below the diagonal, and 304 nonlinear variables.
```

The first two lines repeat information given in the GAMS model statistics and the last two lines describe second order information. CONOPT uses the matrix of second derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian). The Hessian is symmetric and the statistics show that it has 152 elements on the diagonal and 228 below for a total of 380 elements in this case. This compares favorably to the number of elements in the matrix of first derivatives (the Jacobian).

For some models you may see the following message instead (before the usual CONOPT banner):

```
 ** Warning **  Memory Limit for Hessians exceeded.
 You can use the Conopt option "rvhess"
```

The creation of the matrix of second derivatives has been interrupted because the matrix became too dense. A dense matrix of second derivatives will be slow to compute and it will need a lot of memory. In addition, it is likely that a dense Hessian will make some of the computations inside the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

GAMS/CONOPT can use second derivatives even if the Hessian is not available. A special version of the function evaluation routine can compute the Hessian multiplied by a vector (the so-called directional second derivative) without computing the Hessian itself. This routine is used when the Hessian is not available. The directional second derivative approach will require one directional second derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

If you get the "Memory Limit for Hessians exceeded" message you may consider rewriting some equation. Look for nonlinear functions applied to long linear or separable expressions such as log(sum(i,x(i)); as discussed in section 6.3. An expression like this will create a dense Hessian with card(i) rows and columns. You should consider introducing an intermediate variable that is equal to the long linear or separable expression and then apply the nonlinear function to this single variable. You may also experiment with allocating more memory for the dense Hessian and use it despite the higher cost; it may reduce the number of iterations. This can be done by adding the option Rvhess = XX to the CONOPT options file. Rvhess is a memory factor with default value 10 so you need a larger value. The value 0 is special; it means do not impose a memory limit on the Hessian.

The time spend on the many types of function and derivative evaluations are reported in the listing file in a section like this:

```
    CONOPT time Total                         0.734 seconds
      of which: Function evaluations          0.031 =   4.3%
                1st Derivative evaluations    0.020 =   2.7%
                2nd Derivative evaluations    0.113 = 15.4%
                Directional 2nd Derivative    0.016 =   2.1%
```

The function evaluations are computations of the nonlinear terms in the model, and 1st Derivatives evaluations are computations of the Jacobian of the model. 2nd Derivative evaluations are computations of the Hessian of the Lagrangian, and Directional 2nd derivative evaluations are computations of the Hessian multiplied by a vector, computed without computing the Hessian itself. The lines for 2nd derivatives will only be present if CONOPT has used this type of 2nd derivative.

If your model is not likely to benefit from 2nd derivative information or if you know you will run out of memory anyway you can save a small setup cost by telling GAMS/CONOPT not to generate it using option Dohess = f.

A12   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.

A13 Miscellaneous Topics

A13.1 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, e.g., *conopt.opt* and *conopt.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.

A13.2   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, CONOPT and PATH 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 a CNS model with a standard solve statement combined with an options file that has the statement "lssqrs = t". In the latter case, CONOPT will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. The solution process does not include a lot of the safeguards used for ordinary NLP models and when it work it is often very fast and it uses less memory than for the corresponding NLP model. The lack of safeguards means that the solution process just will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Examples of difficulties are that the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds as described with examples below.

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 CONOPT in special cases can guarantee 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 in the current point and there is no unique search direction for Newtons method so CONOPT terminates. 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 CONOPT can guarantee 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.

It should be mentioned that the linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is

```
e1.. x1*x2 =E= 1;
e2.. x1+x2 =E= 3;
x1.l = 1; x2.l = 1;
```

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 will also be combined with model status 5-Locally Infeasible. In the cases where CONOPT can guarantee 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 can be found at the GAMS web site: http://www.gams.com/docs/document.htm.

A13.3   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.

A13.4   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 A13.3 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.

A13.5   External Equations and Extrinsic Functions

CONOPT can be used with external equations and extrinsic functions written in a programming language such as Fortran or C. CONOPT can also use Hessian time vector products from these external libraries. Additional information is available at GAMS's web site at http://www.gams.com/docs/externalequ.htm and in the GAMS User's Guide Appendix J.

Note that CONOPT has a Function and Derivative Debugger. Since external equations and extrinsic functions could contain errors and therefore are dangerous to use CONOPT will automatically turn the Function and Derivative Debugger on in the initial point if the model uses external libraries. After verifying that the external part of the model has been programmed correctly you may turn debugging off again by setting Lkdebg to 0 in an options file.

The debugger has two types of check. The first type ensures that the external equations do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these check are usually caused by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external equations and extrinsic functions are consistent with the rate of change in function values. A derivative is considered to be wrong if the value returned by the modeler deviates from the value computed using numerical differences by more than Rtmxj2 times the step used for the numerical difference (usually around 1.e-7). This check is correct if second derivatives are less than Rtmxj2. Rtmxj2 has a default value of 1.e4. If your model has larger second derivatives you may increase it in order not to get wrong error messages.

The number of error messages from the Function and Derivative Debugger is limited by `Lfderr` with a default value of 10.

9   APPENDIX B - Options

The options that ordinary GAMS users can access are listed below. Options starting on R assume real values, options starting on LS assume logical values (TRUE, T, 1, or FALSE, F, or 0), and all other CR-Cells starting on L assume integer values. The logical option Dohess is only used by the interface between GAMS and CONOPT.

Option	Description	Default
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.	auto
lfilos	Iteration Log frequency for SLP and SQP iterations. A log line is printed to the screen every `lfilos` iterations while using the SLP or SQP mode. The default value depends on the size of the model: it is 1 for large models with more than 2000 constraints or 3000 variables, 5 for medium sized models with more than 500 constraints or 1000 variables, and 10 for smaller models.	auto
lfderr	The Function and Derivative Debugger (by default used with external equations) will not write more than `lfderr` error messages independent of the number of errors found.	10
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.	5
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.	12
lfnsup	Maximum Hessian dimension. If the number of superbasics exceeds `lfnsup` CONOPT will no longer store and use a Reduced Hessian matrix. However, it can still use second derivatives in combination with a conjugate gradient algorithm. The default value depends on the size of the model and its overall memory use. It is usually not a good idea to increase `lfnsup` much beyond its default. The time used to manipulate a very large reduced Hessian matrix is often large compared to the potential reduction in the number of iterations.	auto
lfscal	Frequency for scaling. The scale factors are recomputed after `lfscal` recomputations of the Jacobian.	5
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 A13.4 exceeds `lfstal` and `lfstal` is positive.	100
lkdebg	Controls the Function and Derivative Debugger. The value 0 indicates that the debugger should not be used, the value -1 that it should be used in the initial point, and the value +n that it should be used every n'th time the derivatives are computed. The default value is 0, except for models with external equations where it is -1.	auto

Option	Description	Default
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.	0
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.	false
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.	false
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.	true
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.	true
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.	true
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).	true
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 A13.2. Users are recommended to use the CNS model class in GAMS.	false
lstria	Logical switch for triangular models. If lstria = t (default is f or false) then the model must be triangular as discussed in section A13.1.	false
rtmaxj	Maximum Jacobian element. The optimization is stopped if a Jacobian element (the derivative of a constraints with respect to a variable) exceeds this value and CONOPT issues a "Too large Jacobian element" messages. The default value is 1.e10. If you have very large Jacobian elements then the function values will change very rapidly with small changes in variables and CONOPT may find it difficult to solve the model.	1.e10
rtmaxv	Internal value of infinity. The model is considered unbounded if a variable exceeds rtmaxv in absolute value.	1.e10
rtmaxs	Scale factors larger than rtmaxs are rounded down to rtmaxs. The default value is 2**30 or around 1.07e9. The value is always rounded to a power of 2.	1024**3

Option	Description	Default
rtmxj2	Upper bound on second derivatives. Is used by the Function and Derivative Debugger to test if the difference between a derivative computed by the modeler and a derivative computed from numerical differences is so small that it could be caused by to second order terms. If a model has large second derivatives that causes the Function and Derivative Debugger flags to derivative as incorrect when they are indeed OK then rtmxj2 can be increased. rtmxj2 does not have any influence on the solution algorithm.	1.e4
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.	1.e-5
rtmins	Scale factors smaller than rtmins are rounded up to rtmins. The default value is 1. (All scale factors are powers of 2 to avoid round-off errors from the scaling procedure).	1
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).	1.e-7
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. 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 A13.2) are always solved to the tolerance rtnwmi.	4.e-10
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.	2.e-8
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 value is used in tests for "Slow Progress", see lfnicr.	3.e-13
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.	0.2
rtpiva	Absolute pivot tolerance. A pivot element is only considered acceptable if its absolute value is larger than rtpiva. You may have to decrease this value towards 1.e-11 or 1.e-12 on poorly scaled models.	1.e-10
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. 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.	0.05
rtpivt	Triangular pivot tolerance. A nonlinear triangular pivot element is considered acceptable if its absolute value is larger than rtpivt. Linear triangular pivot must be larger than rtpiva.	1.e-7

Option	Description	Default
rtredg	Optimality tolerance. The reduced gradient is considered zero and the solution optimal if the largest superbasic component is less than rtredg. If you have problems with slow progress or stalling you may increase rtredg. This is especially relevant for very large models.	1.e-7
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.	3
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.	4
dohess	A logical variable that controls the creation of the Hessian (matrix of second derivatives). The default value depends on the model. If the number of equalities is very close to the number of non-fixed variables then the solution is assumed to be in a corner point or in a very low dimensional space where second derivatives are not needed, and dohess is initialized to false. Otherwise dohess is initialized to true. If dohess is false you will not get statistics about the Hessian in the listing file. It takes time to generate second order information and it uses memory. If GAMS generates this information for your model but CONOPT does not use it, i.e. if you see that no time is spend on 2nd derivative evaluations, then you may experiment with dohess turned off. If the number of Hessian elements is very large you may also try turning dohess off. Note that CONOPT still can use directional second derivatives and therefore use its SQP algorithm in the cases where the Hessian is not available.	auto
rvhess	An integer number that controls the space available for creation of the Hessian. Zero is special, meaning no limit on memory for the Hessian. The option also takes real values, but these are currently truncated to integers. With GAMS 24.1, real values will not be truncated to integers anymore.	10

10　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).