Using CONOPT with AMPL

AMPL/CONOPT is an optimization system for nonlinear mathematical programs in continuous variables. It has been developed jointly by AMPL Optimization LLC, responsible for the AMPL/CONOPT interface, and ARKI Consulting & Development A/S, responsible for the CONOPT solution engine. Sales are handled by AMPL Optimization LLC.

This supplement to *AMPL: A Modeling Language for Mathematical Programming* describes the most important features of CONOPT for users of AMPL.

Section 1 describes the kinds of problems to which CONOPT is applicable, and section 2 explains how solver specific options can be passed from AMPL to CONOPT. Section 3 describes the iteration log, section 4 describes the many types of termination messages that CONOPT can produce, and section 5 discusses Function Evaluation Errors. Appendix A gives a short overview over the CONOPT algorithm. Appendix B gives a list of the options that control the interface between AMPL and CONOPT and Appendix C gives a list of the options that control the CONOPT algorithm. The `solve_result_num` values and the corresponding text string in the `solve_message` text is returned by a solve statement are given in Appendix D. References are given in Appendix E.

---

1 Applicability

CONOPT is designed to solve "smooth" nonlinear programs as described in chapter 13 of *AMPL: A Modeling Language for Mathematical Programming* (in the following referred to as The AMPL Book). CONOPT can accommodate both smooth nonlinear functions in the objective and in the constraints. When no objective is specified, CONOPT seeks a point that satisfies the given constraints and thus can be used to solve systems of nonlinear equations and/or inequalities. Non-smooth nonlinearities will also be accepted, but CONOPT is not designed for them; the algorithm is based on the assumption that all functions are smooth and it will in general not produce reliable results when they are not.

CONOPT can also be used to solve linear programs, but other solvers specifically designed for linear programs will in general be more efficient.

CONOPT does not solve integer programs as described in Chapter 15 of the AMPL Book. When presented with an integer program, CONOPT ignores the integrality restrictions on the variables, as indicated by a message such as

```
CONOPT 3.014R: ignoring integrality of 10 variables
```

It then solves the continuous relaxation of the resulting problem, and returns a solution in which some of the integer variables may have fractional values. Solvers that can handle integrality constraints can be found at the AMPL website.

Like most other nonlinear programming solvers CONOPT will attempt to find a local optimum, i.e., a solution that is the best in a neighborhood around the solution. Better solutions may in general exist elsewhere in the solution space. Some models have the property that a local optimum is also a global optimum, i.e., a solution that is the best among all solutions. Among these models are convex optimization models, where the set of feasible solutions is convex and

the objective function is convex (minimization). It is the responsibility of the modeler to ensure that the model is convex, that the model for some other reason only has one local solution, or that the solution produced by CONOPT is an acceptable local solution.

Models with piecewise-linear terms as described in the AMPL Book (chapter 17 of the second edition, 14 of the first) can be used with CONOPT. AMPL will automatically convert the piecewise-linear terms into a sum of linear terms (assuming option `pl_linearize` has its default value 1), send the latter to CONOPT, and convert the solution back; the conversion has the effect of adding a variable to correspond to each linear piece. The piecewise-linear terms are mathematically non-smooth, but AMPL's expansion converts them into a sum of smooth terms that satisfies the smoothness assumptions in CONOPT. If you use option `pl_linearize 0` AMPL will send the non-smooth model to CONOPT and CONOPT will attempt to solve it as if it was smooth; the result is a high likelihood that CONOPT will get stuck in one of the non-smooth points.

Piecewise-linear terms can still give rise to difficulties due to non-convexities. They are only safe provided they satisfy certain convexity rules: Any piecewise-linear term in a minimized objective must be convex, i.e., its slopes must form an increasing sequence as in:

<<-1,1,3,5; -5,-1,0,1.5,3>> x[j]

Any piecewise-linear term in a maximized objective must be concave, i.e., its slopes must form a decreasing sequence as in:

<<1,3; 1.5,0.5,0.25>> x[j]

In constraints, any piecewise-linear term must be either convex and on the left-hand side of a $\leq$ constraint (or equivalently, the right-hand side of a $\geq$ constraint), or else concave and on the left-hand side of a $\geq$ constraint (or equivalently, the right side of a $\leq$ constraint). Piecewise-linear programs that violate the above rules are converted using integer variables, but CONOPT will ignore the integrality and just issue a message like the one shown above and it may put the pieces together in the wrong order and return a solution that does not correspond to AMPL's definition of the piecewise linear term.

Piecewise-linear terms can be used to convert certain non-smooth functions into a form more suitable for CONOPT. Examples are abs, min, and max where abs(x) can be replaced by <<0;-1,1>>x, min(0,x) by <<0;-1,0>>x, and max(0,x) by <<0;0,1>>x.

---

2 Controlling CONOPT from AMPL

In many instances, you can successfully use CONOPT by simply specifying a model and data, setting the `solver` option to `conopt`, and typing `solve`. For larger or more difficult nonlinear models, however, you may need to pass specific options to CONOPT to obtain the desired results. You will also need options to get more detailed messages and iteration output that may help you to diagnose problems.

To give directives to CONOPT, you must first assign an appropriate character string to the AMPL option called `conopt_options`. When `solve` invokes CONOPT, CONOPT breaks this string into a series of individual options. Here is an example:

```
ampl: model pindyck.mod;
ampl: data pindyck.dat;
ampl: option solver conopt;
ampl: option conopt_options 'workmeg=2 \
ampl?   maxiter=2000';
ampl: solve;

CONOPT 3.14R: workmeg=2
maxiter=2000
CONOPT 3.14R: Locally optimal; objective 1170.486285
16 iterations; evals: nf = 35, ng = 0, nc = 58, nJ = 15, nH = 2, nHv = 13
```

All the directives described below have the form of an identifier, an = sign, and a value. You may store any number of concatenated options in `conopt_options`. The example above shows how to type all directives in one long string, using the \ character to indicate that the string continues on the next line. Alternatively, you can list several strings, which AMPL will automatically concatenate:

```
ampl: option conopt_options 'workmeg=2'
ampl?   ' maxiter=2000';
```

In this form, you must take care to supply the space that goes between the options; here we have put it before `maxiter`. Alternatively, when specifying only a few options, you can simply put them all on one line:

```
ampl: option conopt_options 'workmeg=2 maxiter=2000';
```

If you have specified the directive above, and then want to set `maxtime` to `500` you may think to type:

```
ampl: option conopt_options
ampl?   ' maxtime=500';
```

This will replace the previous `conopt_options` string, however; the other previously specified options such as `workmeg` and `maxiter` will revert to their default values. (CONOPT supplies a default value for every option not explicitly specified; the defaults are indicated in the discussion below and in Appendices B and C.) To append new options to `conopt_options`, use this form:

```
ampl: option conopt_options $conopt_options
ampl?   ' maxtime=500';
```

The $ in front of an option name denotes the current value of that option, so this statement appends more options to the current option string. Note the space before `maxtime`.

The available options can be divided into two groups: Interface options are related to the way AMPL and CONOPT communicates with each other; they are described in Appendix B. Algorithmic options control aspects of the CONOPT solution algorithm; they are described in Appendix C.

3 Iteration Output

Following the AMPL conventions CONOPT will by default display very little output. You may get something like:

```
CONOPT 3.14R: Locally optimal; objective 1170.486285
16 iterations; evals: nf = 35, ng = 0, nc = 58, nJ = 15, nH = 2, nHv = 13
```

The first line gives an identification of the solver, a classification of the solution status, and the final objective function value. The second line gives some statistics: number of iterations and number of function calls of various types: nf = number of objective function evaluations, ng = number of gradient evaluations (this particular model has a linear objective function so the gradient is constant and is not evaluated repeatedly), nc = number of constraint evaluations, nJ = number of evaluations of the Jacobian of the constraints, nH = number of evaluations of the Hessian of the Lagrangian function, and nHv = number of evaluations of the Hessian of the Lagrangian multiplied by a vector.

To get more information from CONOPT you must increase the output level, `outlev`. With `outlev=2` general messages from CONOPT (except the iteration log) will go to stdout and the model above may give something like:

```
CONOPT 3.14R: outlev=2

    The model has 112 variables and 97 constraints
    with 332 Jacobian elements, 80 of which are nonlinear.
    The Hessian of the Lagrangian has 32 elements on the diagonal,
    48 elements below the diagonal, and 64 nonlinear variables.

 ** Optimal solution. Reduced gradient less than tolerance.

 CONOPT time Total                            0.141 seconds
   of which: Function evaluations             0.000 =  0.0%
             1st Derivative evaluations       0.000 =  0.0%
             2nd Derivative evaluations       0.000 =  0.0%
             Directional 2nd Derivative       0.000 =  0.0%

CONOPT 3.14R: Locally optimal; objective 1170.486285
16 iterations; evals: nf = 35, ng = 0, nc = 58, nJ = 15, nH = 2, nHv = 13
```

The time spend in various types of function evaluation are listed. Function evaluations correspond to nf + nc, 1st Derivative evaluation correspond to ng + nJ, 2nd Derivative evaluations correspond to nH, and Directional 2nd Derivatives correspond to nHv. On problems that are solved quickly, some times may be reported as zero, as in this example, due to the granularity of the timer.

With `outlev=3` you will also get information about the individual iterations similar to the following:

```
CONOPT 3.14R: outlev=3

    The model has 112 variables and 97 constraints
    with 332 Jacobian elements, 80 of which are nonlinear.
    The Hessian of the Lagrangian has 32 elements on the diagonal,
    48 elements below the diagonal, and 64 nonlinear variables.

 iter phase numinf  suminf/objval   nsuper  rgmax    step   initr mx ok
    0     0         2368.91092
    1     0         14.8848226
    2     0         1.97454016
    3     0      0  0.00414943373                   1            0  1
    4     0      0  1.10604059e-05                  1            0  1
    5     0      0  1.81205579e-08                  1            0  1
    6     3         1150.89301          16  42       0.32       1  0  1
    7     3         1165.06202          16  1e+02    0.31       1  0  1
    8     3         1168.00681          16  28       0.0041        0  1
    9     3         1169.17253          16  12       0.0075        0  1
   10     4         1170.07584          16  11       1             0  1
   11     4         1170.48623          16  1.8      1          6  0  1
   12     4         1170.48628          16  0.021    1          5  0  1
   13     4         1170.48628          16  0.00014  1          4  0  1
   14     4         1170.48629          16  6.3e-07  12            0  1
   15     4         1170.48629          16  5.2e-07  1          1  0  1
   16     4         1170.48629          16  4.8e-08  1          1  0  1

 ** Optimal solution. Reduced gradient less than tolerance.

 CONOPT time Total                            0.031 seconds
   of which: Function evaluations             0.000 =  0.0%
             1st Derivative evaluations       0.000 =  0.0%
             2nd Derivative evaluations       0.000 =  0.0%
             Directional 2nd Derivative       0.000 =  0.0%

CONOPT 3.14R: Locally optimal; objective 1170.486285
16 iterations; evals: nf = 35, ng = 0, nc = 58, nJ = 15, nH = 2, nHv = 13
```

The ten columns in the iterations output have the following meaning (more details on the algorithm can be found in Appendix A):

- Iter: The iteration counter. The first three lines have special meaning: iteration 0 refers to the initial point as received from AMPL, iteration 1 refers to the point obtained after preprocessing, and iteration 2 refers to the same point but after scaling.
- Phase: The remaining iterations are characterized by the Phase in column 2. During Phases 0, 1, and 2 the model is infeasible and an artificial "Sum of infeasibilities" objective function is minimized. During Phases 3 and 4 the model is feasible and the actual Objective function is optimized.
- Numinf: Number of infeasibilities. During Phases 1 and 2 Numinf indicates the number of infeasible constraints. CONOPT adds "Artificial" variables to these constraints and minimizes a "Sum of Infeasibility" objective function; this is similar to the standard phase-1 procedure known from Linear Programming. During this process CONOPT will ensure that the feasible constraints remain feasible. The distinction between Phases 1 and 2 is related to the degree of nonlinearity: during Phase 1 CONOPT will minimize the infeasibilities using a completely linear approximation to the model. During Phase 2 some degree of second order information is included. Phase 1 iterations are therefore cheaper than Phase 2 iterations. Phase 0 indicates a special cheap method for finding a feasible solution: A promising set of basic variables is selected and the basic variables are changed

using a Newton method. Newton's method may not converge quickly if the constraints are very nonlinear; in this case CONOPT removes some of the more nonlinear constraints or constraints with large infeasibilities from the process and tries to satisfy a smaller set of constraints. The number of constraints that has been removed from the Newton process is listed in Numinf and it will most likely grow during Phase 0.

- Suminf/Objval: During Phases 0, 1, and 2 this column will list the sum of absolute values of the infeasibilities, i.e., the artificial objective function that is being minimized. During Phases 3 and 4 this column shows the user's objective function that is being optimized.
- Nsuper: Number of super-basic variables. This number measures the dimension of the space over which CONOPT is optimizing or the degrees of freedom in the model at the current point, taking into account the lower and upper bounds that are active. The second order information that CONOPT uses during Phases 2 and 4 is of dimension Nsuper.
- Rgmax: The absolute value of the largest reduced gradient, taken over the super-basic variables. The optimality conditions require the reduced gradient to be zero so this number is a measure of the non-optimality of the current point.
- Step: The steplength for the iteration. The interpretation depends on several things: During Phase 0 steplength 1 corresponds to a full Newton step. During iterations in which Initr (see next) is positive steplength 1 correspond to full use of the solution from the LP or QP submodel. Finally, iterations in Phase 3 with no value in the Initr column steplength 1 correspond to a full Quasi-Newton step.
- Initr: A number in column eight is a counter for "Inner Iterations". If column eight is empty CONOPT will use steepest descend iterations during Phases 1 and 3 and Quasi-Newton iterations during Phases 2 and 4. Otherwise CONOPT will use an iterative procedure to find a good search direction and Initr indicates the number of iterations in this procedure: During Phases 1 and 3 where CONOPT uses first order information only the feasibility or optimality problem is formulated as an LP model and CONOPT uses an approximate solution to this model as a search direction. Similarly, during Phases 2 and 4 where second order information is used the feasibility or optimality problem is formulated as a QP model and CONOPT uses an approximate solution to this model as a search direction.
- Mx is 1 if the step was limited by a variable reaching a bound and 0 otherwise.
- Ok is 1 if the step was "well behaved" and 0 if the step had to be limited because nonlinearities of the constraints prevented CONOPT from maintaining feasibility.

In the particular log file shown above the cheap Newton-based Phase 0 procedure finds a feasible solution after iteration 5 without introducing any artificial variables. CONOPT continues in Phase 3 which means that the iterations are based on linear information only. From iteration 10 CONOPT switches to Phase 4 which means that it takes into account second order information. In several of the iterations there is a entry in the initr column, indicating that CONOPT performs several inner iterations trying to get a good search direction from a QP-based approximation to the overall model.

---

4 CONOPT Termination Messages

CONOPT may terminate in a number of ways. Standard AMPL messages such as

```
CONOPT 3.14R: Locally optimal; objective 1170.486285
```

have only a few classifications that are sufficient for most purposes (see Appendix D for a list of the solve_message text strings and the associated numerical solve_result_num values). More detailed messages are available from CONOPT and this section will show most of them and explain their meaning. Note that the detailed messages only are available if you use `outlev=2` or `3`. The first 4 messages are used for optimal solutions:

```
** 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.

```
** 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 optimality tolerance `rtredg` with default value around `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 optimality tolerance `rtredg`. However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. The objective must be large or the reduced objective must have large second derivatives for this message to appear so it is advisable to scale the model.

Scaling the model will in this note mean to change the units of variables and equations. You should try to select units for the variables such that the solution values of variables that are not at an active bound are expected to be somewhere between 0.01 to 100. Similarly, you should try to select units for the equations such that the activities described by the equations are somewhere between 0.01 and 100. These two scaling rules should as a by-product result in first derivatives in the model that are not too far from 1.

The last termination message for an optimal solution is:

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

CONOPT stopped 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 a scaling factor equal to the largest Jacobian element divided by 100. Again, the model must have large derivatives for this message to appear so it is advisable to scale the model.

The four messages above all exist in versions where “Optimal” is replaced by “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 and you should try to start CONOPT from a different starting point.

```
** 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 stopped 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. The problem can be caused by discontinuities if the model has discontinuous functions such as abs, min, or max. In this case you should consider alternative, smooth formulations. The problem can also be caused by a poorly scaled model. Finally, it can be caused by stalling as described in section A14 in Appendix A. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". 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.

```
Variable <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. 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 at a solution far from "infinity", CONOPT will have to follow a curved path all the way to "infinity" and it will actually return a feasible solution with very large values for some of the variables. On the way to "infinity" terms in the model may become very large and CONOPT may run into numerical problems and may not be able to maintain feasibility.

The variable names will by default appear as "_svar[xx]" where xx is the internal index in CONOPT. You can request AMPL to pass the proper names to CONOPT by defining

```
option conopt_auxfiles cr;
```

before the `solve` statement in your AMPL program and CONOPT will then use these names in the messages.

The message above exists in a version where "Unbounded" is replaced by "Infeasible". You may also see a message like

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

** Infeasible solution. A free variable exceeds the allowable
   range.  Current value is 1.10E+10 and current upper bound
   (Rtmaxv) is 1.00E+10
```

These messages indicate that some variable became very large before a feasible solution was

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. You can also experiment with alternative starting points; the path towards a feasible point depends heavily on the initial point.

CONOPT may also stop on various resource limits:

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

The time limit defined by option `maxftime` (default value 999999 seconds) has been reached.

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

The iteration limit defined by option `iterlim` (default value 1000000 iterations) has been reached.

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

The number of function evaluation errors has reached the limit defined by option `errlim`. See section 5 for more details on “Function Evaluation Errors”.

The next two messages appear if a derivative (Jacobian element) is very large, either in the initial point or in a later intermediate point.

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

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

and

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

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

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

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. There is also in this case a lazy solution: increase the limit on Jacobian elements, `rtmaxj`; however, you will most likely 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 size statistics provided by AMPL. 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 options like `workfactor=x.x` or `workmeg=x.x`. `Workfactor` will multiply the default estimate by a factor and this is usually the preferred method. `Workmeg` will allocate a fixed amount of memory, measured in Mbytes. The memory related messages you will see are 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
```

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 you will not get any solution value back. If the memory problem appears later during the optimization CONOPT will usually return primal solution values. The marginal values of both equations and variables will be zero.

---

5. Function Evaluation Errors

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

In some cases bounds are not sufficient, e.g., in the expression `log(sum{i in I} x[i])` where each individual `x` should be allowed to become zero, but the sum must be strictly positive. In this case you should introduce an intermediate variable bounded away from zero and an extra equation, e.g.,

```
var xsum >= 0.01;
```

and

```
subject to xsumdef: xsum = sum {i in I} x[i];
```

and use `xsum` as the argument to the `log` function.

Whenever a nonlinear function is called outside its domain of definition, AMPLs function evaluator will intercept the function evaluation error and prevent that the system crashes. AMPL

will report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Many function evaluation errors are usually a sign that something is wrong, so CONOPT stops after `errlim` errors.

During Phases 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 AMPLs 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 `errlim`. The default value is 500.

Even though CONOPT may be able to recover from function evaluation errors it is better to prevent them with bounds on the model variables or with properly bounded intermediate variables.

In some cases function evaluation errors may cause CONOPT to get stuck, for example when there is no previous point to backtrack to, when "bad" points are very close to "reasonable" feasible points, or when 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.

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 AMPL/CONOPT

AMPL/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)
and               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 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 AMPL model is simple: The inequalities defined in the "subject to" section of the AMPL model with =< or => are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all AMPL equality constraints to ensure that the Jacobian matrix, i.e., the matrix of first derivatives of the functions g with respect to the variables x, has full row rank. All these slacks together with the normal AMPL variables are included in x. lo represent the lower bounds defined in AMPL in connection with the var declaration or with the var.lb notation, as well as any bounds on the slacks. Similarly, up represent upper bounds defined in AMPL as well as any bounds on the slacks. g represent the non-constant terms of the AMPL equations themselves; non-constant terms appearing on the right hand side are moved by AMPL 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 AMPL expression to be minimized or maximized.

The AMPL interface has routines for evaluating the nonlinear functions as well as their first and second derivatives. AMPL will also inform CONOPT about sparsety pattern and about which parts of the model are linear and which are nonlinear.

CONOPT assumes that f and g are differentiable with smooth first derivatives. If they are not, e.g., because you have used functions like abs, min, max, round, or trunk, then CONOPT will not know about it. CONOPT will work with an approximation to the real model based on function values and first derivatives and it can easily get stuck around points where any of these quantities are discontinuous.

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 AMPL/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 AMPL 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 will discuss some of the enhancements in CONOPT that make it possible to solve large models efficiently.

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, a scaling procedure, as well as a special Phase 0 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 (the classical “ratio-test”). 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 need 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 sections give some short guidelines for selecting non-default options (section A12) and discuss miscellaneous topics such as numerical difficulties due to loss of feasibility (A13) and slow or no progress due to stalling (A14).

A3. Iteration 0: The Initial Point

The first few “iterations” in the iteration log (see section 3 in the main text for an example) are special initialization iterations. Iteration 0 corresponds to the input point exactly as it was

received from AMPL. The sum of infeasibilities in the column labeled “suminf” includes all residuals.

A4. Iteration 1: Preprocessing

The first part of 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, the equations and variables are ordered as shown in Fig. 1.

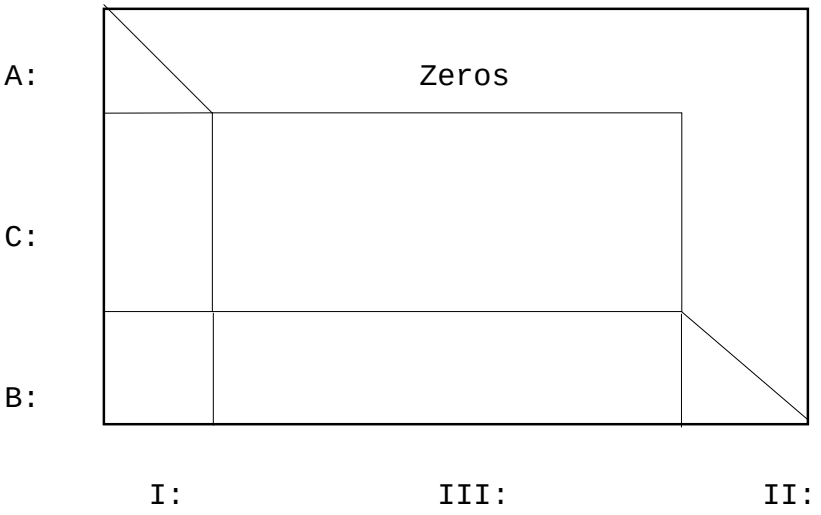

Figure 1: The ordered Jacobian after Preprocessing.

A4.1 Preprocessing: Pre-triangular Variables and Constraints

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

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

the pre-triangular equations are then computed so they become available in AMPL.

You should note that AMPL has a similar preprocessing step. However, AMPL will only solve a variable/constraint pair if the variable appears linearly and the solution therefore is unique. CONOPT will also attempt to solve the variable/constraint pair if the relationship is nonlinear, usually reaching a solution that is close to the initial point provided by the modeler.

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

```
var x1 >= 0.1;
var x2;
var x3;

minimize obj: x1^2 + 2*x2^2 + 3*x3^2;

subject to

e1: log(x1) + x2 = 1.6;
e2: 5*x2 = 3;
```

Equation `e2` is first solved with respect to `x2` (result 3/5 = 0.6). Since the relationship is linear and the solution is unique this is done by AMPL. `x2` is fixed by AMPL, the equation is removed, and CONOPT will not even see it. AMPLs preprocessor will not make further changes. Once CONOPT receives the model equation `e1` is a candidate for preprocessing since `x1` is the only remaining non-fixed variable in the equation. Here `x1` appears nonlinearly and the value of `x1` is found using an iterative scheme based on Newton's method with several safeguards. 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)= 2.00000000E-08

constraint e1: Infeasibility in pre-triangular part of model.
variable x1: Infeasibility in pre-triangular part of model.
```

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`.

Another type of infeasibility is shown by the following model:

```
var x1;
var x2;
var x3 >= 0.1;
var x4;

minimize obj: x1^2 + 2*x2^2 + 3*x3^2 + 4*x4^2;

subject to
```

```
e1: x1^2 + x2 = 1.6;
e2: 5*x2 = 3*x3;
e3: x3^2 = 1;
```

where we have included an extra variable and constraint, `log(x1)` has been replaced by `x1^2`, 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 the pivot is too small.
    Adding a bound or initial value may help.

    Residual=           1.0000000
    Tolerance (Rtnwtr)= 2.00000000E-08

constraint e1: Infeasibility in pre-triangular part of model.
variable x1: Infeasibility in pre-triangular part of model.

The solution order for the critical equations and variables is:

Equation e3 solved with respect to variable x3
Solution value =               1
Equation e2 solved with respect to variable x2
Solution value =             0.6
Equation e1 could not be solved with respect to variable x1
Final solution value =             0
e1 remains infeasible with residual         -1
```

To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility (assuming `outlev>=3`): In this case equation `e3` was first solved with respect to variable `x3`, then equation `e2` was solved with respect to variable `x2` (the linear equation has not been removed by AMPL this time since it depends on `x3` from a nonlinear equation that AMPL did not remove), and finally CONOPT attempted to solve equation `e1` with respect to `x1` at which stage the problem appeared. 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`. To make the analysis easier CONOPT will always report the minimal set of triangular equations and variables that caused the infeasibility so parts of the model that are irrelevant for the reported infeasibility can be ignored.

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

```
var x1 >= 0.1;
var x2;
var x3 >= 0.1;
var x4;

minimize obj: x1^2 + 2*x2^2 + 3*x3^2 + 4*x4^2;

subject to

e1: log(x1) + x2 = 1.6;
e2: 5*x2 = 3*x3;
e3: x3^2 = 1;
e4: x1 + x2 = 3.318;
```

that is derived from the previous models by the addition of a lower bound on `x1` and 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=           1.03684271E-04
    Tolerance (Rtnwtr)= 2.00000000E-08

constraint e1: Inconsistency in pre-triangular part of model.

The solution order for the critical equations and variables is:

Equation e3 solved with respect to variable x3
Solution value =               1
Equation e2 solved with respect to variable x2
Solution value =             0.6
Equation e4 solved with respect to variable x1
Solution value =           2.718
All variables in equations e1 are now fixed
and the equation is infeasible. Residual = -0.000103684
```

First `e3` is solved with respect to `x3`, and `e2` is solved with respect to `x2`. At this point `x1` appear as the only variable in both equation `e1` and `e4` and since `e4` is linear it is selected and `e4` is solved with respect to `x1`. Now all variables that appear in equation `e1`, namely `x1` and `x2`, are fixed, but the equation is not feasible. `e1` is therefore inconsistent with the other constraints as indicated by the first part of the message. In this case the inconsistency is fairly small, 1.03E-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.

You can turn the identification and solution of pre-triangular variables and equations off with the option `lspret=0`. 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:

```
var x1;
var x2;
var x3 >= 0.1;
var x4 >= -1;
var x5;

minimize obj: x1^2 + sqrt(0.01+x2-x4) + 4*x5^2;

subject to

e1: log(1+x1) + x2 = 0;
e2: 5*x2 = -3*x3;
e3: x3^2 = 1;
e4: x4  <= x2;
```

All the nonlinear functions are defined in the initial point in which all variables except `x3` have their default value of zero. The pre-processor will compute `x3=1` from `e3`, `x2=-0.6` from `e2` and `x1=0.822` from `e1`. When CONOPT continues and attempts to evaluate `obj`, the argument to the `sqrt` function is negative when these new solution 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 `obj` 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. CONOPT will only guarantee that simple bounds are strictly enforced at all times. Also note that if the option `lspret=0` 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

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

This part of CONOPT is less important with AMPL since the intermediate variables are much better treated using AMPL's "defined variables," which are described in Appendix A of the AMPL book (§A8.1 of the second edition, §A15 of the first).

A4.3 Preprocessing: The Optional Crash Procedure

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

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a row becomes 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 on the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.

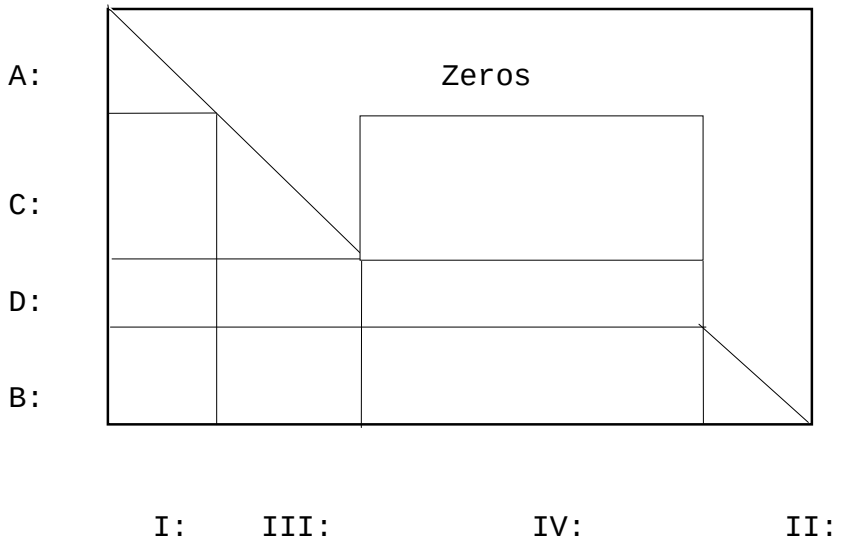

Figure 2: The ordered Jacobian after Preprocessing and Crashing.

Fig. 2 shows a reordered version of fig. 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g., intermediate variables.

The result of the crash procedure is usually an updated initial point in which a large number of equations will be feasible, namely all equations labeled A, B, and C in Fig. 2. There is 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 option `lstcrs=1`. The default value of `lstcrs` (= Logical Switch for Triangular CRaSh) is 0 or false, i.e., the crash procedure is not normally used.

A5. Scaling

After preprocessing the model is scaled, if scaling is turned on, which is the default. The Infeasibility column in the iteration log shows the scaled sum of infeasibilities, except for the initial value in iteration 0 and the value after preprocessing in iteration 1.

Most models are solved faster and more reliably with the automatic scaling procedure turned on. However, it can be difficult to scale models with a mixture of very large and very small variables and/or derivatives so you should in all cases be careful with the selection af units for the variables and equations in your models (see the hints in section 4 in the main text).

The scaling procedure multiplies all variables in group III and all constraints in group C (see Fig. 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 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 around 1.e9 so terms much larger than this will still remain large.
3. The terms 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 variables are divided by this 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 1.e9) are only partially scaled down.

You should be aware of the implications of this scaling procedure: If a constraint has large terms then the accuracy (i.e., the feasibility tolerance) with which it is satisfied is measured relative to the large terms. Depending on the dual variables, the feasibility tolerance is somewhere between rtnwma=1.e-7 and rtnwmi=4.e-10. This means that if you have terms around 1.e9 then the absolute feasibility tolerance will be between 0.4 and 100. On the other hand, if a constraint only has small terms then it is not scaled up and the feasibility tolerance works as an absolute tolerance. So if all terms are around 1.e-5 you may only have a few significant digits.

To avoid difficulties with rapidly varying variables and derivatives CONOPT keeps moving averages of the variables and derivatives and uses these averages instead of the variables and derivatives themselves in the scaling procedure described above. It also recomputes the scale factors at regular intervals (see lfscal). Usually the moving average procedure works well. However, if the size of some variables or derivatives in the initial point deviates by several orders of magnitude from their size in the final point then the scale factors may not catch up with these changes and parts of the solution may become inaccurate. Initial points with very large derivatives, e.g., 1/x started at x = 1.e-6, should therefore be avoided.

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 AMPL. This point will always satisfy the bounds (3): the AMPL/CONOPT driver will simply move a variable that is outside its bounds to the nearest bound before it is presented to CONOPT. 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), in which case CONOPT must first find an initial feasible point. For models without an objective function, feasibility is the only problem. Finding a feasible point can be just as hard as finding an optimum in problems that have an objective function.

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 Phases 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; however, sometime is does not 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, 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, 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) 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) 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 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, AMPL/CONOPT has found a feasible solution and it will proceed to Phases 3 and 4. Otherwise, Phases 1 and 2 are used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: iter, phase, numinf, suminf, step, mx, and ok. The number in the numinf column counts the number of "difficult" infeasible equations, and the number in the suminf column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. Suminf will decrease while numinf may increase. An iteration with a full Newton step will be represented by step = 1, mx = 0 and ok = 1. An iteration with a step determined by a basic variable reaching a bound will be represented by mx = 1 and ok = 1; a positive step value represents the fraction of the Newton step that was possible, and a zero step indicates a degenerate iteration. And an iteration in which suminf could not be reduced due to nonlinearities will be represented by step = 0, mx = 0, and ok = 0.

The success of the Phase 0 procedure is based on being able to choose a good basis that ideally will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so CONOPT has many variables away from their bounds to choose from.

In addition to the crash option described in §A4.3, the start and the iterations of Phase 0 can be controlled by three options, `lslack`, `lsmxbs`, and `lmmxsf`, which are described in Appendix B.

A7. Finding a Feasible Solution: Phase 1 and 2

All but numinf equations are feasible when phase 0 ends. 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 "difficult" equations), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variables 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 and the original bounds are reestablished. The number of infeasibilities (shown in the numinf column of the iteration log) and the sum of infeasibilities (in the suminf 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 Phase 3 and 4 that is described in the next sections.

A8. Linear and Nonlinear Mode: Phases 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in section 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 steps 4 and 5 are also performed sparsely.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model appears to be 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 approximations for the steps that are taken, then CONOPT takes advantages of the linearity: The derivatives (the components of the Jacobian matrix) are not computed every iteration. Instead, 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 descent algorithm, and the initial step length 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 or Phase 3. Phase 1 means that the model is infeasible and objective is the sum of infeasibilities and Phase 3 means that 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 steps 2 to 9 of the GRG algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed every 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 or Phase 4: during Phase 2 the solution is infeasible, and during Phase 4 it is feasible. The iterations in Phases 2 and 4 are in general more expensive than the iteration in Phases 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 several iterations in Phase 2 or 4 (nonlinear mode). Phases 2 and 4 indicate 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, or is uses explicit 2nd derivatives computed by AMPL.

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 descent 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.

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 option `lseslp=0`. The default value of `lseslp` (= Logical Switch Enabling SLP mode) is 1 (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 descent 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 updating methods for the edge lengths borrowed from LP, but it must re-initialize the edge lengths more frequently, e.g., when an inversion fails, which happens more frequently for nonlinear models than for linear models, especially for 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 option `lsanrm=1`. The default value of `lsanrm` (= Logical Switch for A-NoRM) is `0` (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 Phases 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 will need some kind of second order information to make good progress. The second order information can be estimated over many iterations using Quasi-Newton updating methods like BFGS. However, CONOPT can also use exact second order information about the functions and this information is delivered by AMPL. The second order information is used in a Sequential Quadratic Programming (SQP) procedure that much like the SLP procedure described above finds a good search direction and a good basis; 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 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. This number is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

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 option `lsesqp=0`. The default value of `lsesqp` (= Logical Switch Enabling SQP mode) is `1` (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.

The AMPL/CONOPT interface can generate two types of 2^{nd} order information: the matrix of 2^{nd} derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian), or the Lagrangian of the Hessian multiplied by a vector, also called directional 2^{nd} derivatives. The generation is controlled by the interface option `hess`. By default both types of

2nd derivatives are generated corresponding to `hess=3`. You can ask for only the Hessian as a sparse matrix (`hess=1`) or only the directional 2nd derivatives (`hess=2`) or no 2nd derivatives at all (`hess=0`).

If the interface generates 2nd derivatives as a sparse matrix the log file will a have few extra lines:

```
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 give information about the size of the model and first order information and the last two lines describe second order information. The Hessian of the Lagrangian 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) so it should be cheap to use the Hessian for this model.

In some cases you will find that the Hessian has a large number of elements. This can make the generation and use of the Hessian very expensive and memory intensive. In this case you should consider only to use directional 2nd derivative without computing the Hessian itself (`hess=2`). The directional 2nd derivative approach will require one directional 2nd derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

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 some setup costs by using option `hess=0`.

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 spent in Phase 0, Phases 1 and 3, or in Phases 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 spent in Phase 0 may still be rather large. If this is the case or if the infeasibilities after Phase 0 are large, i.e., many “difficult” equations are excluded from the Newton process then you may try to use the triangular crash options:

```
lstcrs=1
```

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 also try:

```
lslack=1
```

CONOPT will add artificial variables to all constraints that remain infeasible after the preprocessor, 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, but the SLP procedure may pay off.

You may also try the experimental bending line search with

```
lmmxsf=1
```

With this option, the line search in Phase 0 will be different and the infeasibilities may be reduced faster than with the default `lmmxsf=0`. This option may be combined with the triangular crash, `lstcrs=1`. Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

Phases 1 and 3: The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, `lsanrm=1`, but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but there are large variations. Try it when the number of iterations is very large.

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, then it may be better to turn SLP off with `lseslp=0`.

Phases 2 and 4: There are currently not many options available when most of the time is spent in Phase 2 and Phase 4. You can try to change `lfnsup` as discussed in Appendix C. If the change in objective during the final iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, `rtredg`.

A13. 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 marginal values.

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. If this happens with your model you are encouraged to (1) consider if bounds on some degenerate variables can be changed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, `rtnwma` and `rtnwmi` (usually by tightening `rtnwma`).

A14. 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.

Due to small infeasibilities and round-off errors, 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 marginal values (see §A13, where this noise term is also 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-monotonic 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. 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`.

APPENDIX B: Options for the AMPL/CONOPT interface

The interface options are listed in the table below. "int" indicates an integer value and "fp" a floating point value. You can use an e-exponent with floating points, but the Fortran d-exponent is illegal. All non-default options should be included as `option=value` pairs in `conopt_options`.

Keyword	Type	Meaning
`errlim`	int	Limit on function evaluation errors (default = 500). If the objective or constraints cannot be evaluated at a proposed next iterate, CONOPT will try a shorter step at most `errlim` times.
`hess`	int	Whether to use the Hessian of the Lagrangian: 0 = no 1 = use explicit Hessian (explicit 2nd derivatives) 2 = use Hessian-vector products (directional 2nd derivatives) 3 = use both explicit Hessian and Hessian-vector products (default).
`iterlim`	int	Iteration limit (default 1000000).
`logfreq`	int	For `outlev=3` print one summary line every `logfreq` iterations (default = 1).
`maxftime`	fp	Limit on cpu seconds (default = 999999).
`maxfwd`	int	Use forward automatic differentiation for defined variables that depend on at most `maxfwd` other variables (default = 5)
`maximize`	none	Maximize the objective (even if the model says to minimize it).
`maxiter`	int	Iteration limit (default 1000000). Synonym for `iterlim`.
`minimize`	none	Minimize the objective (even if the model says to maximize it).
`objno`	int	Which objective to optimize: 1 = first objective (default), 2 = second objective, etc. 0 = just satisfy the constraints.
`outlev`	int	0 = no options echoed on stdout. 1 = options but no iteration log (default). 2 = CONOPT "screen" output on stdout. 3 = as 2 plus a log line every `logfreq` iterations.
`superbasics`	int	Limit on number of superbasic variables for which a Reduced Hessian is estimated and stored. (Default >= 500).
`timing`	int	0 = no timing report. 1 = timing report on stdout. 2 = timing report on stderr. 3 = timing report on both.
`version`	none	Show version and CONOPT banner.
`workfactor`	fp	If `workfactor` > 1 then multiply the standard memory estimate by this factor.
`workmeg`	fp	0: Let CONOPT decide how much memory to allocate (default). > 0: means allocate `workmeg` megabytes of memory.

APPENDIX C: Options Controlling the CONOPT Algorithm

The CONOPT options that an AMPL user can access are listed below. Options starting on R assume floating point values, options starting on LS assume logical values (use `1` for true and `0` for false), and all other options starting on L assume integer values. Floating point values can be written with or without a decimal point and with or without an e-exponent. The Fortran type d-exponents are not allowed. All non-default options should be included as `option=value` pairs in `conopt_options`.

`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 interface option `outlev`.

`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

`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. The limit is `lfmxns` if `lfmxns` is positive and the square root of the number of structural variables if `lfmxns` is zero. The default value is zero.

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

`lfnsup` Same as interface option `superbasics`. Maximum Hessian dimension. If the number of superbasics exceeds `lfnsup` CONOPT will no longer store a Reduced Hessian matrix. However, it can still use second derivatives in combination with a conjugate gradient algorithm. The default value of `lfnsup` is at least `500` and it is larger for very large models. If you experience many iterations in Phases 2 or 4 with a large number of superbasic variables you may try to increase `lfnsup` to a value above the number of superbasics, but it is not always a good idea to increase `lfnsup` much beyond its default value unless you have a very large model. The time used to manipulate a very large reduced Hessian matrix can be large compared to the potential reduction in the number of iterations. If `lfnsup` is increased the default memory allocation may not be sufficient, and you may have to include a `workfactor` or `workmeg` interface option.

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

`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 A14 exceeds `lfstal`. The default value of `lfstal` is `100`.

`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. However, feasible linear constraints can become infeasible. The alternative method is turned on by setting `lmmxsf` to `1`, and it is turned off by setting `lmmxsf` to `0`. The method is by default turned off.

`lslack` Logical switch for slack basis. If `lslack=1` 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=1` if the number of infeasible constraints after the crash procedure is small. This is especially true if the SLP procedure described in section A9 quickly can remove these remaining infeasibilities. It is necessary to experiment with the model to determine if this option is useful.

`lsmxbs` Logical Switch for Maximal Basis. `lsmxbs` determines whether CONOPT should try to improve the condition number of the initial basis (`1`) before starting the Phase 0 iterations or just use the initial basis immediately (`0`). The default value is `1`, i.e., CONOPT tries to improve the basis. There is a computational cost associated with the procedure, but there will usually be a net savings, 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 `1`.

`lspost` Logical switch for the Post-triangular preprocessor. If `lspost=0` (default is `1`) then the post-triangular preprocessor discussed in section A4.2 is turned off.

`lspret` Logical switch for the Pre-triangular preprocessor. If `lspret=0` (default is `1`) then the pre-triangular preprocessor discussed in section A4.1 is turned off.

`lsscal` A logical switch that turns scaling on (with the value `1`) or off (with the value `0`). The default value is `1`, i.e., scaling is turned on.

`lstcrs` A logical switch that turns the triangular crash (see section A4.3) on (with the value `1`) or off (with the value `0`). The default value is `0`, i.e., the triangular crash is turned off.

`rtmaxj` Maximum Jacobian element. The optimization is stopped if a Jacobian element exceeds this value. The default value of `rtmaxj` is `1.e10`. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve.

`rtmaxv` Maximum size for a variable. The model is considered unbounded if a variable exceeds `rtmaxv` in absolute value. The default value of `rtmaxv` is `1.e10`. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve. Explicit bounds on variables are not allowed to be larger than `rtmaxv`.

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

around 1.e9 (to avoid rounding it is a power of 2, namely `1024^3`).

`rtmins` Scale factors smaller than `rtmins` are rounded up to `rtmins`. The default value is `1`, i.e., CONOPT does by default not round small terms up.

`rtnwma` Maximum feasibility tolerance. A constraint will only be considered feasible if the residual is less than `rtnwma` times MaxJac, independent on the dual variable. MaxJac is an overall scaling measure for the constraints computed as max(1,maximal Jacobian element/100). The default value is `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. The default value is `4.e-10`. You should only increase this number if you have inaccurate function values and you get an infeasible solution with a very small sum of infeasibility, or if you have very large terms in some of your constraints (in which case scaling may be more appropriate).

`rtnwtr` Triangular feasibility tolerance. If you solve a model, fix some of the variables at their optimal value and solve again and the model then is reported infeasible in the pre-triangular part, then you should increase `rtnwtr`. The infeasibilities in some unimportant constraints in the "Optimal" solution have been larger than `rtnwtr`. The default value is `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 default value of `rtobjr` is `3.e-13`. The value is used in tests for "Slow Progress"; see `lfnicr`.

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

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

`rtpivr` Relative pivot tolerance. A pivot element is only considered acceptable relative to other elements in the column if its absolute value is at least `rtpivr` * the largest absolute value in the column. The default value is `0.05`. You may have to increase this value towards 1 on poorly scaled models. Increasing `rtpivr` will result in denser L and U factors of the basis, i.e., you may also have to allocate more memory.

`rtpivt` Triangular pivot tolerance. A nonlinear triangular pivot element is considered acceptable if its absolute value is larger than `rtpivt`. The default value is `1.e-5`. Linear triangular pivots must be larger than `rtpiva`.

`rtredg` Optimality tolerance. The reduced gradient is considered zero and the solution optimal if the largest superbasic component is less than `rtredg`. The default value is `1.e-7`. If you have problems with slow progress or stalling you may increase `rtredg`. This is especially relevant for very large models.

`rvstlm` Step length multiplier. The step length in the one-dimensional line search is not

allowed to increased by a factor of more than `rvstlm` between steps for models with nonlinear constraints and a factor of 100 * `rvstlm` for models with linear constraints. The default value is `4`.

APPENDIX D: Solve_result_num values

In addition to the solution to the model, AMPL's "solve" command will also return a "solve_result_num" value that characterizes the solution. The values along with the text that appears in the associated solve_message are shown in the table below. Values larger than the ones shown here indicate some kind of bug.

Value	Message
0	Optimal.
100	Locally optimal.
101	Bogus "Termination by solver" (CONOPT bug).
200	Infeasible.
201	Locally infeasible.
202	Intermediate infeasible.
300	Unbounded.
400	Iteration limit.
401	Time limit.
500	CONOPT bug: unknown Model Status
501	Intermediate non-optimal.
504	Unknown type of error
505	Error no solution.
506	Evaluation error.
507	Evaluation error limit.
508	objno = nnn is not >=0 and <= nnn.
521	Not enough memory for CONOPT's initial allocations.
522	Too many constraints.
523	Too many variables.
524	Too many variables + constraints.
525	Too many nonzeros.
526	Size permitted by demo license exceeded.
529	CONOPT ran out of memory.

APPENDIX E: 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).

R. Fourer, D.M. Gay, and B.W. Kernighan, AMPL: *A Modeling Language for Mathematical Programming*, The Scientific Press, (2003 = 2nd edition; 1st = 1993).

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).