GAMS/MINOS

Table of Contents:

1. INTRODUCTION

This section describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver. GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems.

GAMS/MINOS is designed to find solutions that are *locally optimal*. The nonlinear functions in a problem must be *smooth* (i.e., their first derivatives must exist). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a *global optimum*. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a staring point that is "sufficiently close," but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this chapter.

2. HOW TO RUN A MODEL WITH GAMS/MINOS

MINOS is capable of solving models of the following types: LP, NLP and RMINLP. If MINOS is not specified as the default LP, NLP or RMINLP solver, then the following statement can be used in your GAMS model:

```
option lp=minos5;   { or nlp or rminlp }
```

It should appear before the `solve` statement.

3. OVERVIEW OF GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

Minimize	$F(x)$	$+ c^T x$	$+ d^T y$		*(1)*
subject to	$f(x)$		$+ A_1 y$	$<> b_1$	*(2)*
		$A_2 x$	$+ A_3 y$	$<> b_2$	*(3)*
	l	$\leq$	x,y	$\leq u$	*(4)*

where the vectors c, d, b_1, b_2, l, u and the matrices A_1, A_2, A_3 are constant, $F(x)$ is a smooth scalar function, and $f(x)$ is a vector of smooth functions. The $< >$ signs mean that individual constraints may be defined using $\leq$, $=$ or $\geq$ corresponding to the GAMS constructs =L= , =E= and =G=.

The components of x are called the nonlinear variables, and the components of y are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let m_1 and n_1 denote the number of nonlinear constraints and variables, and let m and n denote the total number of (general) constraints and variables. Thus, A_3 has m-m_1 rows and n-n_1 columns.

The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of l and u may be $-\infty$ or $+\infty$ respectively, in accordance with the GAMS use of -INF and + INF.

The vectors b_1 and b_2 are called the right-hand side, and together are denoted by b.

3.1. Linear programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program.* Since there is no need to distinguish between linear and nonlinear variables, we use x rather than y. GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\begin{array}{ll} \textit{minimize} & c^T x \\ \textit{subject to} & Ax + Is = 0 \\ & l \leq x,\ s \leq u \end{array}$$

where the elements of x are your own GAMS variables, and s is a set of *slack variables:* one for each general constraint. For computational reasons, the right-hand side b is incorporated into the bounds on s.

In the expression $Ax + Is = 0$ we write the identity matrix explicitly if we are concerned with columns of the associated matrix (AI). Otherwise we will use the equivalent notation $Ax + s = 0$.

GAMS/MINOS solves linear programs using a reliable implementation of the *primal simplex method* (Dantzig, 1963), in which the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Nx_N = 0,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B and x_N are called the basic or nonbasic variables respectively. Together they are a permutation of the vector *(x, s)*.

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equation $Bx_B = Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form.

The simplex method reaches such a solution by performing a sequence of *iterations,* in which one column of B is replaced by one column of *N*(and vice versa), until no such interchange can be found that will reduce the value of c^Tx.

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of x_B lie significantly outside their bounds, we say that the current point is *infeasible*. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function c^Tx.

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly *between* their bounds" $l_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $l_j - \delta < x_j < l_j$ or $u_j < x_j + \delta$ where δ is a *feasibility tolerance* (typically 10^{-6}). In rare cases, even a new nonbasic variables might lie outside their bounds by as much as δ.

GAMS/MINOS maintains a sparse LU factorization of the basis matrix B, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL(Gill *et al.* 1987).(see Bartels and Golub, 1969; Bartels, 1971; Reid, 1976 and 1982.) The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

3.2. Problems with a Nonlinear Objective

When nonlinearities are confined to the term F(x) in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a *reduced-gradient* algorithm (Wolfe, 1962) combined with a *quasi-Newton* algorithm follows that described in Murtagh and Saunders (1978).

In the reduced-gradient method, the constraints Ax + Is = 0 are partitioned into the form

$$Bx_B + Sx_S + Nx_N = 0$$

where x_s is a set of *superbasic variables.* At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance δ), while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be *s*, the number of columns in *S*. (The context will always distinguish s from the vector of slack variables .) At a solution, *s* will be no more than n_1, the number of nonlinear variables. In many practical cases we have found that s remains reasonably small, say 200 or less, even if n_1 is large.

In the reduced-gradient algorithm, x_s is regarded as a set of "independent variables" or "free variables" that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of *B*, *S* and *N*, some of the nonbasic variables are selected to be added to *S*, and the process is repeated with an increased value of *s*. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of *s* is reduced by one.

A step of the reduced-gradient method is called a *minor iteration.* For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix *Z* is needed now for descriptive purposes. It takes the form

$$Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}$$

though it is never computed explicitly. Given LU factorization of the basis matrix *B*, it is possible to compute products of the form Zq and Z^Tg by solving linear equations involving *B* or B^T. This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the *B*, *S*, *N* partition remains constant. A *search direction q* for the superbasic variables is obtained by solving a system of the form

$$R^TRq = -Z^Tg$$

where *g* is a gradient of *F(x)*, Z^Tg is the *reduced gradient*, and *R* is a dense upper triangular matrix. GAMS computes the gradient vector *g* analytically, using symbolic differentiation. The matrix *R* is updated in various ways in order to approximate the *reduced Hessian* according to $R^TR \approx Z^THZ$ where *H* is the matrix of second derivatives of *F(x)* (the *Hessian).*

Once *q* is available, the search direction for all variables is defined by $p = Zq$. A *line search* is then performed to find an approximate solution to the one-dimensional problem

$$\begin{array}{ll} \textit{Minimize}_{\alpha} & F(x + \alpha p) \\ \textit{subject to} & 0 < \alpha < \beta \end{array}$$

where β is determined by the bounds on the variables. Another important GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length α (see Gill *et*

at., 1979). The number of nonlinear function evaluations required may be influenced by setting the Line search tolerance, as discussed in Section D.3.

As a linear programming, an equation $B^T\pi = gB$ is solved to obtain the *dual variables* or *shadow prices* π where *gB* is the gradient of the objective function associated with basic variables. It follows that $gB - B^T\pi = 0$ The analogous quantity for superbasic variables is the reduced-gradient vector $Z^Tg = gs - s^T\pi$; this should also be zero at an optimal solution. (In practice its components will be of order $r \,||\, \pi \,||$ where *r* is the optimality tolerance, typically 10^{-6}, and $||\, \pi \,||$ is a measure of the size of the elements of π.)

3.3. Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a *project Lagrangian* algorithm, based on a method due to Robinson (1972), see Murtagh and Saunders (1982). This involves a sequence of *major iterations*, each of which requires the solution of a *linearly constrained subproblem*. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the *k*-th major iteration, let x_k be an estimate of the nonlinear variables, and let λ_k be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing *f(x)* in equation (2) to its linear approximation:

$$f'(x, x_k) = f(x_k) + J(x_k)\,(x - x_k)$$

or more briefly

$$f' = f_k + J_k\,(x - x_k)$$

where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k. (The *i*-th row of the Jacobian is the gradient vector of the *i*-th nonlinear constraint function. As for the objective gradient, GAMS calculates the Jacobian using symbolic differentiation).

The subproblem to be solved during the *k*-th major iteration is then

$$\begin{array}{lll} \textit{Minimize}_{x,y} & F(x) + c^Tx + d^Ty - \lambda_k^T(f - f') + 0.5\,\rho\,(f - f')^T(f - f') & (5) \\ \textit{subject to} & f' \quad + A_1y \;\; <> b_1 & (6) \\ & A_2x \quad + A_3y \;\; <> b_2 & (7) \\ & l \;<\; (x,y) \;< u & (8) \end{array}$$

The objective function (5) is called an *augmented Lagrangian*. The scalar ρ is a *penalty parameter*, and the term involving ρ is a modified *quadratic penalty function*.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) - (8). As before, slack variables are introduced and b_1 and b_2 are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}+\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} s_1 \\ s_2 \end{pmatrix}=\begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}$$

This system will be referred to as $Ax + Is = 0$ as in the linear case. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_1, A_2, and A_3.

In the output from GAMS/MINOS, the term *Feasible subproblem* indicates that the *linearized constraints* have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that *feasibility* and *optimality* occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved" problems, the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iteration.

4. GAMS OPTIONS

The following GAMS options are used by GAMS/MINOS:

4.1. Options specified through the option statement

The following options are specified through the option statement. For example,

```
set iterlim = 100 ;
```

sets the iterations limit to 100.

`Iterlim`	integer	Sets the iteration limit. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. *(default = 1000)*
`Reslim`	real	Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. *(default = 1000.0)*
`Bratio`	real	Determines whether or not to use an advanced basis. *(default = 0.25)*
`Sysout`	on/off	Will echo the MINOS messages to the GAMS listing file. *(default = off)*

4.2. Options specified through model suffixes

The following options are specified through the use of the model suffix. For example,

```
mymodel.workspace = 10 ;
```

sets the amount of memory used to 10 MB. `Mymodel` is the name of the model as specified by the model statement. In order to be effective, the assignment of the model suffix should be made between the model and solve statements.

`workspace`	real	Gives MINOS x MB of workspace. Overrides the memory estimation.*(default is estimated by solver)*
`optfile`	integer	Instructs MINOS to read the option file *`minos5.opt`*. *(default = 0)*
`scaleopt`	integer	Instructs MINOS to use scaling information passed by GAMS through the *variable.SCALE* parameters*(default = 0)*

5. SUMMARY OF MINOS OPTIONS

The performance of GAMS/MINOS is controlled by a number of parameters or “options.” Each option has a default value that should be appropriate for most problems. (The defaults are given in the Section 7.) For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file.

All these options should be entered in the option file 'minos5.opt' after setting the model.OPTFILE parameter to 1. The option file is not case sensitive and the keywords must be given in full. Examples for using the option file can be found at the end of this section. The second column in the tables below contains the section where more detailed information can be obtained about the corresponding option in the first column.

5.1. Output related options

Debug level	Controls amounts of output information.
Log Frequency	Frequency of iteration log information.
Print level	Amount of output information.
Scale, print	Causes printing of the row and column-scales.
Solution No/Yes	Controls printing of final solution.
Summary frequency	Controls information in summary file.

5.2. Options affecting Tolerances

Crash tolerance	crash tolerance
Feasibility tolerance	Variable feasibility tolerance for linear constraints.
Line search tolerance	Accuracy of step length location during line search.
LU factor tolerance	Tolerance during LU factorization.
LU update tolerance	
LU Singularity tolerance	
Optimality tolerance	Optimality tolerance.
Pivot Tolerance	Prevents singularity.
Row Tolerance	Accuracy of nonlinear constraint satisfaction at optimum.
Subspace tolerance	Controls the extent to which optimization is confined to the current set of basic and superbasic variables

5.3. Options affecting Iteration limits

Iterations limit	Maximum number of minor iterations allowed
Major iterations	Maximum number of major iterations allowed.
Minor iterations	Maximum number of minor iterations allowed between successive linearizations of the nonlinear constraints.

5.4. Other algorithmic options

Check frequency	frequency of linear constraint satisfaction test.
Completion	accuracy level of sub-problem solution.
Crash option	perform crash
Damping parameter	See Major Damping Parameter
Expand frequency	Part of anti-cycling procedure
Factorization frequency	Maximum number of basis changes between factorizations.
Hessian dimension	Dimension of reduced Hessian matrix
Lagrangian	Determines linearized sub-problem objective function.
Major damping parameter	Forces stability between subproblem solutions.
Minor damping parameter	Limits the change in x during a line search.
Multiple price	Pricing strategy
Partial Price	Level of partial pricing.
Penalty Parameter	Value of ρ in the modified augmented Lagrangian.
Radius of convergence	Determines when ρ will be reduced.
Scale option	Level of scaling done on the model.
Start assigned nonlinears	Affects the starting strategy during cold start.
Superbasics limit	Limits storage allocated for superbasic variables.
Unbounded objective value	Detects unboundedness in nonlinear problems.

Unbounded step size	Detects unboundedness in nonlinear problems.
Verify option	Finite-difference check on the gradients
Weight on linear objective	Invokes the composite objective technique,

5.5. Examples of GAMS/MINOS option file

The following example illustrates the use of certain options that might be helpful for "difficult" models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

```
BEGIN GAMS/MINOS options
* These options might be relevant for very nonlinear models.
Major damping parameter  0.2   * may prevent divergence.
Minor damping parameter  0.2   * if there are singularities
                               * in the nonlinear functions.
Penalty parameter       10.0   * or 100.0 perhaps-a value
                               * higher than the default.
Scale linear variables         * (This is the default.)
END GAMS/MINOS options
```

Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the "cautious" defaults are relaxed:

```
BEGIN GAMS/MINOS options
* Suggestions for models with MILDLY nonlinear constraints
Completion   Full
Minor alteration limit   200
Penalty parameter           0.0  * or 0.1 perhaps-a value
                                 * smaller than the default.
                                 *   Scale one of the following
Scale all variables              * if starting point is VERY GOOD.
Scale linear variables           * if they need it.
Scale   No   * otherwise.
END GAMS/MINOS options
```

Most of the options described in the next section should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

6. Special Notes

6.1. Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:

- Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix .L).
- Include sensible upper and lower bounds on all variables.
- Specify a *Major damping parameter* that is lower than the default value, if the problem is suspected of being highly nonlinear.
- Specify a *Penalty parameter* ρ that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values $\lambda_k = 0$ and $\rho = 0$ for all subproblems, by specifying *Lagrangian=No*. However, convergence is much more like with the default setting, *Lagrangian=Yes*. The initial estimate of the Lagrange multipliers is then $\lambda_0 = 0$, but for later subproblems λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is $\rho = 100.0/m_1$ where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many times it is safe to specify $\rho = 0$, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

6.2. Storage

GAMS/MINOS uses one large array of main storage for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of main storage available on a particular machine, and the amount of computation time that one‘s budget and/or patience can stand.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is m, the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly 100*m* words, where one “word” is the relevant storage unit for the floating-point arithmetic being used. This usually means about 800*m* bytes for workspace. A further

300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with m general constraints requires about (*m*+300) K bytes of memory.

Another important quantity, is n, the total number of variables in *x* and *y*. The above comments assume that *n* is not much larger than *m*, the number of constraints. A typical ratio is *n/m* is 2 or 3.

If there are many nonlinear variables (i.e., if n_1 is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects s, the number of superbasic variables. Recall that *s* is zero for purely linear problems. We know that *s* is need never be larger than n_1+1. In practice, *s* is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix *R* has dimension *s* and requires about $½ s^2$ words of storage. If it seems likely that *s* will be very large, some aggregation or reformulation of the problem should be considered.

7. The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section., a sample log file is shown for for each case, and the appearing messages are explained.

7.1. Linear Programs

MINOS uses a standard two-phase Simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem [MEXLS] is as follows:

```
--- Starting compilation
--- mexls.gms(1242)
--- Starting execution
--- mexls.gms(1241)
--- Generating model ONE
--- mexls.gms(1242)
---    353 rows, 578 columns, and 1873 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .21 Mb
 Reading data...

   Itn  Nopt  Ninf  Sinf,Objective
     1     1    74  1.62609255E+01
    20    14    74  1.62609255E+01
```

```
   40     4    67  1.40246211E+01
   60    14    55  1.10065392E+01
   80    12    42  8.35651175E+00

  Itn  Nopt  Ninf  Sinf,Objective
  100     6    31  7.21650054E+00
  120    21    20  6.65998885E+00
  140    13    11  5.77113823E+00
  160     1     8  2.35537706E+00
  180     4     5  9.58810903E-01

  Itn  Nopt  Ninf  Sinf,Objective
  200     9     2  3.16184199E-01

 Itn   208 -- Feasible solution.  Objective =   4.283788540E+04

  220     2     0  4.11169434E+04
  240    25     0  3.88351313E+04
  260     1     0  3.77495457E+04
  280    11     0  3.66399964E+04

  Itn  Nopt  Ninf  Sinf,Objective
  300     2     0  3.30955215E+04
  320     2     0  3.09919632E+04
  340     5     0  2.84521363E+04
  360     1     0  2.76671551E+04
  380     2     0  2.76055456E+04
 Itn   392 --     80  nonbasics set on bound, basics recomputed

 Itn   392 -- Feasible solution.  Objective =   2.756959890E+04


 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              1         392
 Objective function       2.7569598897150E+04
 Degenerate steps              83       21.17
 Norm X,      Norm PI    2.10E+01    4.49E+04
 Norm X,      Norm PI    8.05E+03    1.06E+02  (unscaled)
--- Restarting execution
--- mexls.gms(1242)
--- Reading solution for model ONE
--- All done
```

Considering each section of the log file above in order:

```
----------------------------------------------------------------------
 Work space allocated           --     .21 Mb
 Reading data...
----------------------------------------------------------------------
```

When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem is then loaded into MINOS.

Next, the MINOS iteration log begins. In the beginning, a feasible solution has not yet been found, and the screen log during this phase of the search looks like:

```
------------------------------------ -------------------------
   Itn  Nopt  Ninf  Sinf,Objective
     1     1    74  1.62609255E+01
------------------------------------------------------------------
```

The first column gives the iteration count. A simplex iteration corresponds to a basis change. *Nopt* means "Number of non-optimalities" which refers to the number of marginals that do not have the correct sign (i.e. they do not satisfy the optimality conditions), and *Ninf* means "Number of infeasibilities" which refers to the number of infeasible constraints . The last column gives the "Sum of infeasibilities" in phase 1 (when *Ninf > 0*) or the Objective in phase 2.

As soon as MINOS finds a feasible solution, it returns a message similar to the one below that appears for the example discussed:

```
------------------------------------------------------------------
 Itn   208 -- Feasible solution.  Objective =   4.283788540E+04
------------------------------------------------------------------
```

MINOS found a feasible solution and will switch from phase 1 to phase 2. During phase 2 (or the search once feasibility has been reached), MINOS returns a message that looks like:

```
------------------------------------------------------------------
 Itn   392 --     80  nonbasics set on bound, basics recomputed
------------------------------------------------------------------
```

Even non-basic variables may not be exactly on their bounds during the optimization phase. MINOS uses a sliding tolerance scheme to fight stalling as a result of degeneracy. Before it terminates it wants to get the "real" solution. The non-basic variables are therefore reset to their bounds and the basic variables are recomputed to satisfy the constraints $Ax = b$. If the resulting solution is infeasible or non-optimal, MINOS will continue iterating. If necessary, this process is repeated one more time. Fortunately, optimality is usually confirmed very soon after the first "reset". In the example discussed, feasibility was maintained after this excercise, and the resulting log looks like:

```
------------------------------------------------------------------
 Itn   392 -- Feasible solution.  Objective =   2.756959890E+04
------------------------------------------------------------------
```

MINOS has found the optimal solution, and then exits with the following summary:

```
------------------------------------------------------------------
 Major, Minor itns              1         392
 Objective function       2.7569598897150E+04
 Degenerate steps              83       21.17
 Norm X,      Norm PI    2.10E+01    4.49E+04
 Norm X,      Norm PI    8.05E+03    1.06E+02  (unscaled)
------------------------------------------------------------------
```

For an LP, the number of major iterations is always one. The number of minor iterations is the number of Simplex iterations.

Degenerate steps are ones where a Simplex iteration did not do much other than a basis change: two variables changed from basic to non-basic and vice versa, but none of the activity levels changed. In this example there were 83 degenerate iterations, or 21.17% of the total 392 iterations.

The norms ||*x*|| , ||*pi*|| are also printed both before and after the model is unscaled. Ideally the scaled ||*x*|| should be about 1 (say in the range 0.1 to 100), and ||pi|| should be greater than 1. If the scaled ||*x*|| is closer to 1 than the final ||*x*|| , then scaling was probably helpful. Otherwise, the model might solve more efficiently without scaling.

If ||*x*|| or ||*pi*|| are several orders of magnitude different before and after scaling, you may be able to help MINOS by scaling certain sets of variables or constraints yourself (for example, by choosing different units for variables whose final values are very large.)

7.2. Nonlinear Objective

Consider the WEAPONS model from the model library which has nonlinear objective and linear constraints. The screen output resulting from running this model is as follows:

```
--- Starting compilation
--- weapons.gms(59)
--- Starting execution
--- weapons.gms(57)
--- Generating model WAR
--- weapons.gms(59)
---    13 rows, 66 columns, and 156 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .06 Mb
 Reading data...
 Reading nonlinear code...

   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
     1    -1     3  9.75000000E+01     3   64

 Itn     6 -- Feasible solution.  Objective =   1.511313490E+03

    20     5     0  1.70385909E+03    23   45
    40     3     0  1.73088173E+03    63   30
    60     1     0  1.73381961E+03   109   25
    80     1     0  1.73518225E+03   155   20

   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
   100     0     0  1.73555465E+03   201   17

 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              1         119
 Objective function       1.7355695798562E+03
 FUNOBJ, FUNCON calls         245           0
```

```
 Superbasics, Norm RG          18    2.18E-08
 Degenerate steps               2        1.68
 Norm X,      Norm PI    2.74E+02    1.00E+00
 Norm X,      Norm PI    2.74E+02    1.00E+00  (unscaled)
--- Restarting execution
--- weapons.gms(59)
--- Reading solution for model WAR
--- All done
```

The parts described in the LP example will not be repeated - only the new additions will be discussed.

```
---------------------------------------------------------------------
 Reading nonlinear code...
---------------------------------------------------------------------
```

Besides reading the matrix GAMS/MINOS also reads the instructions that allow the interpreter to calculate nonlinear function values and their derivatives.

The iteration log looks slightly different from the earlier case, and has a couple of additional columns.

```
---------------------------------------------------------------------
   Itn  Nopt  Ninf  Sinf,Objective  Nobj  NSB
     1    -1     3  9.75000000E+01    3   64
---------------------------------------------------------------------
```

The column *Nobj* gives the number of times MINOS calculated the objective and its gradient. The column *NSB* contains the number of superbasics. In MINOS, variables are typed basic, non-basic or superbasic. The number of superbasics once the number settles down (there may be rather more at the beginning of a run) is a measure of the nonlinearity of the problem. The final number of superbasics will never exceed the number of nonlinear variables, and in practice will often be much less (in most cases less than a few hundred).

There is not much that a modeler can do in this regard, but it is worth remembering that MINOS works with a dense triangular matrix *R* (which is used to approximate the reduced Hessian) of that dimension. Optimization time and memory requirements will be significant if there are more than two or three hundred superbasics.

Once the optimum solution has been found, MINOS exits after printing a search summary that is slightly different from the earlier case.

```
---------------------------------------------------------------------
 Major, Minor itns              1         119
---------------------------------------------------------------------
```

As in the earlier case, there is always only one major iteration. Each iteration of the reduced-gradient method is called a minor iteration.

```
---------------------------------------------------------------------
 FUNOBJ, FUNCON calls         245           0
---------------------------------------------------------------------
```

FUNOBJ is the number of times MINOS asked the interpreter to evaluate the objective function and its gradients. *FUNCON* means the same for the nonlinear constraints. As this model has only linear constraints, the number is zero.

If you know that your model has linear constraints but *FUNCON* turns out to be positive, your nonlinear objective function is being regarded as a nonlinear constraint. You can improve MINOS's efficiency by following some easy rules. Please see "Objective: special treatment of in NLP" in the index for the GAMS User's Guide.

```
------------------------------------------------------------------
 Superbasics, Norm RG          18    2.18E-08
------------------------------------------------------------------
```

The number of superbasic variables in the final solution is reported and also the norm of the reduced gradient (which should be close to zero for an optimal solution).

7.3. Nonlinear constraints

For models with nonlinear constraints the log is more complicated. CAMCGE from the model library is such an example, and the screen output resulting from running it is shown below:

```
--- Starting compilation
--- camcge.gms(444)
--- Starting execution
--- camcge.gms(435)
--- Generating model CAMCGE
--- camcge.gms(444)
---    243 rows, 280 columns, and 1356 non-zeroes.
--- Executing MINOS5
 Work space allocated           --     .36 Mb
 Reading data...
 Reading nonlinear code...

 Major Minor  Ninf Sinf,Objective    Viol    RG    NSB  Ncon Penalty    Step
 1     0     1  0.000000E+00   1.3E+02 0.0E+00  184     3 6.8E-01 1.0E+00
 2    40T   30  0.00000000E+00 8.8E+01 7.3E+02  145     4 6.8E-01 1.0E+00
 3    40T   30  0.00000000E+00 8.8E+01 1.7E+01  106     5 6.8E-01 1.0E+00
 4    40T   30  0.00000000E+00 8.8E+01 2.1E+01   66     6 6.8E-01 1.0E+00
 5    40T   25  0.00000000E+00 8.8E+01 1.7E+01   26     7 6.8E-01 1.0E+00
 6    29     0  1.91734197E+02 2.6E-03 0.0E+00    0     9 6.8E-01 1.0E+00
 7     0     0  1.91734624E+02 2.3E-08 0.0E+00    0    10 1.4E+00 1.0E+00
 8     0     0  1.91734624E+02 4.5E-13 0.0E+00    0    11 1.4E+00 1.0E+00
 9     0     0  1.91734624E+02 6.8E-13 0.0E+00    0    12 0.0E+00 1.0E+00

 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns              9         189
 Objective function       1.9173462423688E+02
 FUNOBJ, FUNCON calls           8          12
 Superbasics, Norm RG           0    0.00E+00
 Degenerate steps               0         .00
```

```
 Norm X,      Norm PI    1.37E+03    2.88E+01
 Norm X,      Norm PI    8.89E+02    7.01E+01  (unscaled)
 Constraint violation    6.82E-13    7.66E-16
--- Restarting execution
--- camcge.gms(444)
--- Reading solution for model CAMCGE
--- All done
```

Note that the iteration log is different from the two cases discussed above. A small section of this log is shown below.

```
----------------------------------------------------------------------
Major Minor  Ninf  Sinf,Objective    Viol    RG    NSB  Ncon Penalty Step
1     0      1  0.00000000E+00 1.3E+02 0.0E+00  184     3 6.8E-01 1.0E+00
2    40T   30  0.00000000E+00 8.8E+01 7.3E+02  145     4 6.8E-01 1.0E+00
----------------------------------------------------------------------
```

Two sets of iterations - Major and Minor, are now reported. A description of the various columns present in this log file follows:

Major	A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.
Minor	The number of minor iterations performed on the linearized subproblem. If it is a simple number like 29, then the subproblem was solved to optimality. Here, *40T* means that the subproblem was terminated. (There is a limit on the number of minor iterations per major iteration, and by default this is 40.) in general the *T* is not something to worry about. Other possible flags are *I* and *U*, which mean that the subproblem was Infeasible or Unbounded. MINOS may have difficulty if these keep occurring.
Ninf	The number of infeasibilities at the end of solving the subproblem. It is 0 if the linearized constraints were satisfied. It then means that all linear constraints in the original problem were satisfied, but only *Viol* (below) tells us about the nonlinear constraints.
Objective	The objective function for the original nonlinear program.
Viol	The maximum violation of the nonlinear constraints.
RG	The reduced gradient for the linearized subproblem. If *Viol* and *RG* are small and the subproblem was not terminated, the original problem has almost been solved.
NSB	The number of superbasics. If *ninf > 0* at the beginning, it will not matter too much if *NSB* is fairly large. It is only when *ninf* reaches 0 that MINOS starts working with a dense matrix *R* of dimension *NSB*.

Ncon — The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

Penalty — The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

Step — The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. The quantity *Viol* should then decrease rapidly. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the *Major Damping parameter*.

The final exit summary has an additional line in this case. For the problem being described, this line looks like:

```
------------------------------------------------------------------
 Constraint violation    6.82E-13    7.66E-16
------------------------------------------------------------------
```

The first number is the largest violation of any of the nonlinear constraints (the final value of *Viol*). The second number is the relative violation, $Viol/(1.0 + ||x||)$. It may be more meaningful if the solution vector x is very large.

For an optimal solution, these numbers must be small.

Note: The CAMCGE model (like many CGE models or other almost square systems) can better be solved with the MINOS option *Start Assigned Nonlinears Basic*. It can be noticed in the log that the algorithm starts out with lots of super basics (corresponding to the non-linear variables). MINOS starts with removing these superbasics, and only then starts working towards an optimal solution.

---

8. Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. The letters I and r denote integer and real values. The number ε denotes machine precision (typically 10^{-15} or 10^{-16}). Options not specified will take the default values shown.

Check frequency i — Every *i*-th iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = 0$ where s is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is

computed. If the largest component of *r* is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

(*Default = 30*)

Completion Full/Partial

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion), or to full accuracy (full completion), GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems.

Use of partial completion may reduce the work during early major iterations, unless the *Minor iterations* limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion.

An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below *100*(Row tolerance)*, the relative change in λ_k is 0.1 or less, and the previous subproblem was solved to optimality.

Full completion tends to give better Langrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

(*Default = FULL*)

Crash option i

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix (*AI*). The value of *i* determines which columns of A are eligible. Columns of *I* are used to fill “gaps” where necessary.

If *i > 0*, three passes are made through the relevant columns of *A*, searching for a basis matrix that is essentially triangular. A column is assigned to “pivot” on a particular row if the column contains a suitably large element in a row that has not yet been assigned.(The pivot elements ultimately form the diagonals of the triangular basis).

Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality (=E=) constraints. Pass 3 allows the pivots to be in inequality rows.

For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

(*Default = 1*)

0	The initial basis will contain only slack variables: $B = I$
1	All columns of A are considered (except those excluded by the *Start assigned nonlinears* option).
2	Only the columns of A corresponding to the linear variables y will be considered.
3	Variables that appear nonlinearly in the objective will be excluded from the initial basis.
4	Variables that appear nonlinearly in the constraints will be excluded from the initial basis.

Crash tolerance r

The *Crash tolerance r* allows the starting procedure *CRASH* to ignore certain "small" nonzeros in each column of *A*. If a_{max} is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|aij| < a_{max}.r$. To be meaningful, *r* should be in the range $0 \leq r < 1$).

When r > 0.0 the basis obtained by *CRASH* may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of *A* and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A are the matrix shown under *LU factor tolerance* ; i.e., a tridiagonal matrix entries -1, 4, -1. To help *CRASH* choose all m columns for the initial basis, we could specify *Crash tolerance r* for some value of $r > 0.25$.

(*Default = 0.1*)

Damping parameter r

See Major Damping Parameter

(*Default = 2.0*)

Debug level i

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness.

(*Default = 0*)

0	No debug output.
2 (or more)	Output from *M5SETX* showing the maximum residual after a row check.
40	Output from *LU8RPC* (which updates the LU factors of the

basis matrix), showing the position of the last nonzero in the transformed incoming column

50 Output from *LU1MAR* (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.

100 Out from *M2BFAC* and *M5LOG* listing the basic and superbasic variables and their values at every iteration.

Expand frequency i This option is part of anti-cycling procedure designed to guarantee progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is δ.

Over a period of i iterations, the tolerance actually used by GAMS/MINOS increases from 0.5 δ to δ (in steps $0.58\ \delta / i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing *i* helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see *Pivot tolerance*).

(*Default = 10000*)

Factorization frequency i At most i basis changes will occur between factori-zations of the basis matrix.

With linear programs, the basis factors are usually updated every iteration. The default i is reasonable for typical problems. Higher values up to i =100 (say) may be more efficient on problems that are extremely sparse and well scaled.

When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made

regularly (according to the *Check frequency*) to ensure that the general constraints are satisfied. If necessary the basis will be re-factorized before the limit of i updates is reached.

When the constraints are nonlinear, the Minor iterations limit will probably preempt i.

(*Default = 50*)

Feasibility tolerance r

When the constraints are linear, *a feasible solution* is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance *r*. (Since slacks are included, this means that the general linear constraints are also satisfied to within r.)

GAMS/MINOS attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let *SINF* be the corresponding sum of infeasibilities, If *SINF* is quite small, it may be appropriate to raise *r* by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

If *SINF* is not small, there may be other points that have a significantly smaller sum of infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum.

If *Scale option = 1* or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful)

A nonlinear objective function *F(x)* will be evaluated only at feasible points. If there are regions where *F(x)* is undefined, every attempt should be made to eliminate these regions from the problem. For example, for a function *F(x) = sqrt(x_1) + log(x_2)*, it should be essential to place lower bounds on both variables. If *Feasibility tolerance = 10^{-6}*, the bounds $x_1 > 10^{-5}$ and $x_2 > 10^{-4}$ might be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.)

If the constraints are nonlinear, the above comments apply to each major iteration. A “feasible solution” satisfies the current linearization of the constraints to within the tolerance *r*. The associated subproblem is said to be feasible.

As for the objective function, bounds should be used to keep *x* more than *r* away from singularities in the constraint functions *f(x)*.

At the start of major iteration *k*, the constraint functions $f(x_k)$

are evaluated at a certain point x_k. This point always satisfies the relevant bounds ($l < x_k < u$), but may not satisfy the general linear constraints.

During the associated minor iterations, *F(x)* and *f(x)* will be evaluated only at points x that satisfy the bound and the general linear constraints (as well as the linearized nonlinear constraints).

If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily, in several stages.

Feasibility with respect to the nonlinear constraints themselves is measured against the *Row tolerance* (not against *r*. The relevant test is made at the *start* of a major iteration.

(*Default = 1.0E-6*)

Hessian dimension r

This specifies than an *r X r* triangular matrix *R* is to be available for use by the quasi-Newton algorithm (to approximate the reduced Hessian matrix according to $Z^THZ \approx R^TR$. Suppose there are s superbasic variables at a particular iteration. *Whenever possible, r should be greater than s.*

If $r > s$, the first *s* columns of *R* will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear.

If $r < s$, a matrix of the form,

$$R = \begin{pmatrix} R_r & 0 \\ & D \end{pmatrix}$$

will be used to approximate the reduced Hessian, Where R_r is an *r X r* upper triangular matrix and *D* is a *diagonal* matrix of order *s - r*. The rate of convergence will no longer be superlinear (and may be arbitrarily slow.

The storage required if of the order $½ r^2$, which is substantial if *r* is as large as 200 (say). In general, *r* should be slight over-estimate of the final number of superbasic variables, whenever storage permits. It need not be larger than n_1+1, where n_1 is the number of nonlinear variables. For many problems it can be much smaller than n_1.

If *Superbasics limit s* is specified, the default value of *r* is the same number, *s* (and conversely). This is a safeguard to

ensure super-linear convergence wherever possible. If neither *r* nor *s* is specified, GAMS chooses values for both, using certain characteristics of the problem.

(*Default = Superbasics limit*)

Iterations limit i

This is maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the *GAMS ITERLIM* specification. If i = 0, no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

Iters or *Itns* are alternative keywords.

(*Default = 1000*)

Lagrangian Yes/No

This determines the form of the objective function used for the linearized subproblems. The default value *yes* is highly recommended. The *Penalty parameter* value is then also relevant.

If *No* is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not occur.

(*Default = yes*)

Linesearch tolerance r

For nonlinear problems, this controls the accuracy with which a step-length α is located in the one-dimensional problem

$$\begin{array}{ll} \textit{Minimize}_{\alpha} & F(x + \alpha p) \\ \textit{subject to} & 0 < \alpha \leq \beta \end{array}$$

A linesearch occurs on most minor iterations for which *x* is feasible. [If the constraints are nonlinear, the function being minimized is the augmented Lagrangian in equation (5).]

r must be a real value in the range $0.0 < r < 1.0$.

The default value $r = 0.1$ requests a moderately accurate search. It should be satisfactory in most cases.

If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try $r = 0.01$ or $r = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints.

If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try *r = 0.5* or perhaps *r = 0.9*. (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

(*Default = 0.1*)

Log Frequency i

In general, one line of the iteration log is printed every *i*-th minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions,

A value such as *i = 10, 100* or larger is suggested for those interested only in the final solution

Log frequency 0 may be used as shorthand for *Log frequency 99999*.

If *Print level > 0*, the default value of *i* is 1. If *Print level = 0*, the default value of *i* is 100. If *Print level = 0* and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

(*Default = 1 or 100*)

LU factor tolerance r_1
LU update tolerance r_2
LU Singularity tolerance r_3

The first two tolerances affect the stability and sparsity of the basis factorization $B = LU$ during re-factorization and updates respectively. The values specified must satisfy $r_i \geq 1.0$. The matrix L is a product of matrices of the form.

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix}$$

where the multipliers μ will satisfy $|\mu| < r_i$.

1. The default values $r_i = 10.0$ usually strike a good compromise between stability and sparsity.
2. For large and relatively dense problems, $r_i = 25.0$ (say) may give a useful improvement it sparsity without impairing stability to a serious degree.
3. For certain very regular structures (e.g., band matrices) it may be necessary to set and r_1 and/or r_2 to values smaller than the default in order to achieve stability. For example, if

the columns of *A* include a sub-matrix of the form

$$\begin{pmatrix} 4 & -1 & \\ -1 & 4 & -1 \\ & -1 & 4 \end{pmatrix}$$

it would be judicious to set both r_1 and r_2 to values in the range $1.0 < r_i < 4.0$. The singularity tolerance r_3 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \le r_3$ or $|U_{jj}| < r_3 \, max_i \, |U_{jj}|$, the j-th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases , the Jacobian matrix may converge to values that make the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting $r_3 = 1.0E\text{-}5$, say, may help cause a judicious change of basis.

(*Default values:* $r_1 = 10.0$, $r_2 = 10.0$, $r3 = \varepsilon^{2/3} \approx 10^{-11}$)

Major damping parameter r

The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions (x_k, λ_k) and (x_{k+1}, λ_{k+1}). For example, the default value 2.0 prevents the relative change in either x_k or λ_k from exceeding 200 percent. It will not be active on well behaved problems.

The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus x_{k+1} and λ_{k+1} are changed to

$$x_k + \sigma(x_{k+1} - x_k) \text{ and } \lambda_k + \sigma(\lambda_{k+1} - \lambda_k)$$

for some step-length $\sigma < 1$. In the case of nonlinear equation (where the number of constraints is the same as the number of variables) this gives a *damped Newton method*.

This is very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher *Penalty parameter* (say 10 or 100 times the default ρ). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of ρ is irrelevant.)

If the subproblem solutions continue to change violently, try reducing *r* to 0.2 or 0.1 (say).

For implementation reason, the shortened step to σ applies to the nonlinear variables x, but not to the linear variables *y* or the slack variables *s*. This may reduce the efficiency of the control.

(*Default = 2.0*)

Major iterations i

This is maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations my not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

(*Default = 50*)

Minor damping parameter r

This parameter limits the change in *x* during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

A linesearch of the form $minimize_{\alpha}\ F(x + \alpha p)$ is performed over the range $0 < \alpha \leq \beta$, where β is the step to the nearest upper or lower bound on *x*. Normally, the first step length tried is $\alpha_1 = min\ (1,\beta)$.

In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of *r* can lean to floating-point overflow. The parameter *r* is therefore used to define a limit

$$\beta' = r\ (1 + ||x||)/||p||$$

and the first evaluation of *F(x)* is at the potentially smaller steplength $\alpha_1 = min(1, \beta, \beta')$.

Wherever possible, upper and lower bounds on *x* should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value *r = 2.0* shout not affect progress on well behaved problems, but setting *r = 0.1 or 0.01* may be helpful when rapidly varying function are present. A "good" starting point may be required. An important application is to the class of nonlinear least squares problems.

In case where several local optima exist, specifying a small value for *r* may help locate an optima near the starting point.

(*Default = 2.0*)

Minor iterations i

This is the maximum number of minor iterations allowed between successive linearizations of the nonlinear constraints.

A moderate value (e.g., $20 \leq i \leq 50$) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion.

The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations, (say *K*) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify > *K*, to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. (If a "good" initial subproblem is arbitrarily interrupted by a small I th subsequent linearization may be less favorable than the first.)

In general it is unsafe to specify value as small as *i* = 1or 2.)even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal.)

The *Iteration limit* provides an independent limit on the total minor iterations (across all subproblems).

If the constraints are linear, only the *Iteration limit* applies: the *minor iterations* value is ignored.

(*Default = 40*)

Multiple price i

"pricing" refers to a scan of the current non-basic variables to determine if any should be changed from their value (by allowing them to become superbasic or basic).

If multiple pricing in effect, the *i* best non-basic variables are selected for admission of appropriate sign.) If partial pricing is also in effect , the best *i* best variables are selected from the current partition of *A* and *I*.

The default *i = 1* is best for linear programs, since an optimal solution will have zero superbasic variables.

Warning : If *i > 1*, GAMS/MINOS will use the *reduced-gradient method* (rather than the simplex method) even on purely linear problems. The subsequent iterations of *not* correspond to the efficient "minor iterations" carried out be commercial linear programming system using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving *i* dense columns of dimension *m*, and *i* is therefore limited for storage reasons

typically to the range $2 \leq i \leq 7$.)

GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of i . Larger values of i are therefore practical, but in general the iterations and time required when $i > 1$ are greater than when the simplex method is used ($i=1$) .

On large nonlinear problems it may be important to set $i > 1$ if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use $i = 10$ (say) for early runs, until it seems that the number of superbasics has leveled off.

If *Multiple price i* is specified , it is also necessary to specify *Superbasic limit s* for some $s > i$.

(*Default = 1*)

Optimality tolerance r

This is used to judge the size of the reduced gradients $d_j = g_j - \pi^T a_j$, where g_j is the gradient of the objective function corresponding to the j-th variable. a_j is the associated column of the constraint matrix (or Jacobian), and π is the set of dual variables.

By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy

$$dj / ||\pi|| \geq -r \quad or \quad dj / ||\pi|| \leq r$$

respectively, and if $dj / ||\pi|| \leq r$ for superbasic variables.

In the $||\pi||$ is a measure of the size of the dual variables. It is included to make the tests independents of a scale factor on the objective function.

The quantity actually used is defined by

$$\sum_{i=1} \quad , \qquad \| \pi \| = max\{ \sigma / \sqrt{m}, 1 \}$$

so that only large scale factors are allowed for.

If the objective is scaled down to be *small*, the optimality test effectively reduced to comparing D_j against r.

(*Default = 1.0E-6*)

Partial Price i

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (when a nonbasic variable is selected to become basic or superbasic).

When $i = 1$, all columns of the constraints matrix (A I) are searched.

Otherwise, A_j and I are partitioned to give i roughly equal segments A_j , I_j $(j = 1$ *to* $i)$. If the previous search was successful on A_{j-1} , I_{j-1}, the next search begins on the segments A_j , I_j. (All subscripts here are modulo i.)

If a reduced gradient is found that is large than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (several may be selected if multiple pricing has been specified.) IF nothing is found, the search continues on the next segments $A_j + 1$, $I_j + 1$ and so on.

Partial price t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

(*Default = 10 for LPs, or 1 for NLPs*)

Penalty Parameter r

This specifies the value of ρ in the modified augmented Lagrangian. It is used only when *Lagrangian = yes* (the default setting).

For early runs on a problem is known to be unknown characteristics, the default value should be acceptable. If the problem is problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of ρ may be necessary of known to ensure convergence, *even convex programs.*

On the other hand, if ρ is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify *penalty parameter 0.0*

Initially, use a moderate value for r (such as the default) and a reasonably low *Iterations* and/or *major iterations* limit.

If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the *Major damping parameter*.)

If there appears to be little progress between major iteration, it may help to reduce the penalty parameter.

(*Default = 100.0/*m_1)

Pivot Tolerance r

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular. The default value of *r* should be satisfactory in most circumstances.

When x changes to $x + \alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.

For linear problems, elements of P are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.

For nonlinear problems, elements smaller than $r\ ||p||$ are ignored.

It is common (on "degenerate" problems) for two or more variables to reach a bound at essentially the same time. In such cases, the *Feasibility tolerance* (say *t*) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of *t* should not be specified.

To a lesser extent, the *Expand frequency* (say *f*) also provides some freedom to maximize pivot the element. Excessively large values of *f* should therefore not be specified .

(*Default* = $\varepsilon^{2/3} \approx 10^{-11}$)

Print level i

This varies the amount of information that will be output during optimization.

Print level 0 sets the default *Log* and *summary frequencies* to 100. It is then easy to monitor the progress of run.

Print level 1 (or more) sets the default *Log* and *summary frequencies* to 1, giving a line of output for every minor iteration. *Print level 1* also produces basis statistics., i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized.

For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The value of i is best thought of as a binary number of the form

Print level *JFLXB*

where each letter stand for a digit that is either 0 or 1. The quantities referred to are:

B — Basis statistics, as mentioned above.

X — x_k, the nonlinear variables involved in the objective function or the constraints.

L — λ_k, the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if *Lagrangian=No*, since then $\lambda_k=0$.)

F — $f(x_k)$, the values of the nonlinear constraint functions.

J — $J(x_k)$, the Jacobian matrix.

To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, *Print level 10* sets *X=1* and the other digits equal to zero; the nonlinear *variables* will be printed each major iteration.

If *J =1*, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if *J = 1*, there is no reason to specify *X = 1* unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3   1.250000D+01   BS   1   1.00000D+00
4   2.00000D+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually different from the GAMS row numbers; see the *Solution* option.)

(*Default = 0*)

Radius of convergence r

This determines when the penalty parameter ρ will be reduced (if initialized to a positive value). Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multiplier estimate must be less than *r* at the start of a major iteration before ρ is reduced or set to zero.

A few major iterations later, full completion will be requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

(*Default = 0.01*)

Row Tolerance r

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about that an accuracy.

Let *ROWERR* be the maximum component of the residual vector $f(x) + A_1y - b_1$, normalized by the size of the solution. Thus

$$ROWERR = || f(x) + A_1y - b_1||_\infty / (1+XNORM)$$

Where *XNORM* is a measure of the size of the current solution (x,y). The solution is regarded acceptably feasible if $ROWERR \leq r$.

If the problem functions involve data that is known to be of low accuracy, a larger *Row tolerance* may appropriate.

(*Default = 1.0E-6*)

Scale option i

Scaling done on the model.

(*Default = 2 for LPs, 1 for NLPs*)

0

No scaling. If storage is at a premium, this option should be used

1

Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer, 1982). This will sometimes improve the performance of the solution procedures. *Scale linear variables* is an equivalent option.

2

All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side *b* or the solution *x* is large. This takes into account columns of (*AI*) that are fixed or have positive lower bounds or negative upper bounds. *Scale nonlinear variables* or *Scale all variables* are equivalent options.

Scale Yes sets the default. (*Caution*: If all variables are nonlinear, *Scale Yes* unexpectedly does nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent to *Scale Option 0*).

If nonlinear constraints are present, *Scale option 1* or *0* should generally be rid at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should therefore be used only if (a) good starting point is provided, and (b) the problem is not highly nonlinear.

Scale, print

This causes the row-scales *r(i)* and column-scales *c(j)* to be printed. The scaled matrix coefficients are $a'_{ij} = a_{ij}\, c(j)/r(i)$, and the scaled bounds on the variables, and slacks are $l'_j = l_j/c(j)$, $u'_j = u_j/c(j)$, where $c(j) = r(j-n)$ if $j > n$.

If a *Scale option* has not already been specified, *Scale, print* sets the default scaling.

Scale Tolerance r

All forms except *Scale option* may specify a tolerance *r* where $0 < r < 1$ (for example: *Scale, Print, Tolerance = 0.99*). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ration of the largest and smallest nonzero coefficients in each column:

$$\rho_j = max_i\, |a_{ij}| \,/\, min_i\, |a_{ij}| \;\; (a_{ij} \neq 0)$$

If $max_j\ \rho_j$ is less than *r* times its previous value, another scaling pass is performed to adjust the row and column scales. Raising *r* from 0.9 to 0.99 (say) usually increases the number of scaling passes through *A*. At most 10 passes are made.

If a *Scale option* has not already been specified, *Scale tolerance* sets the default scaling.

(*Default = 0.9*)

Solution No/Yes

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS "column numbers," and those at the right of each line in the rows section are GAMS/MINOS "slacks".)

The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

(*Default = No*)

Start assigned nonlinears

This option affects the starting strategy when there is no basis (i.e., for the first solve or when *option bratio = 1* is used to reject an existing basis.)

This option applies to all nonlinear variables that have been assigned non-default initial values and are strictly between their bounds. Free variables at their default value of zero are

excluded. Let *K* denote the number of such "assigned nonlinear variables."

Note that the *first* and *fourth* keywords are significant.

(*Default = superbasic*)

Superbasic — Specify *superbasic* for highly nonlinear models, as long as *K* is not too large (say *K < 100)* and the initial values are "good".

Basic — Specify *basic* for models that are essentially "square" (i.e., if there are about as many general constraints as variables).

Nonbasic — Specify *nonbasic* if *K* is large.

Eligible for crash — Specify *eligible for Crash* for linear or nearly linear models. The nonlinear variables will be treated in the manner described under *Crash* option.

Subspace tolerance r — This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3).

r must be a real number in the range $0 < r \leq 1$.

When a nonbasic variables x_i is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $||Z^T g_0||$. (In fact, the norm will be $|d_i|$, the size of the reduced gradient for the new superbasic variable x_j .

Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies $\|Z^T g_0\| \leq r$ $\|Z^T g_0\|$ is the size of the largest reduced-gradient component among the superbasic variables.)

A smaller value of *r* is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as *r* = 0.9 may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution.

Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of *r*.

(*Default = 0.5*)

Summary frequency i — A brief form of the iteration log is output to the summary

file. In general, one line is output every *i*-th minor iteration., In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated.

The *Summary frequency* controls summary output in the same as the *log frequency* controls output to the print file

A value such as *i* = 10 or 100 is often adequate to determine if the SOLVE is making progress. If *Print level = 0*, the default value of *i* is 100. If *Print level > 0*, the default value of *i* is 1. If *Print level = 0* and the constraints are nonlinear, the *Summary frequency* is ignored. Instead, one line is printed at the beginning of each major iteration.

(*Default = 1 or 100*)

Superbasics limit i

This places a limit on the storage allocated for superbasic variables. Ideally, *i* should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.

For linear problems, an optimum is normally a basic solution with no degrees of freedom.(The number of variables lying strictly between their bounds is not more than *m*, the number of general constraints.) The default value of *i* is therefore 1.

For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables."

Normally, *i* need not be greater than n_1+1, where n_1 is the number of nonlinear variables.

For many problems, *i* may be considerably smaller than n_1. This will save storage if n_1 is very large.

This parameter also sets the *Hessian dimension*, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

(*Default = Hessian dimension*)

Unbounded objective value r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If $|F|$ exceeds r or if α exceeds r_2, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`.

If singularities are present, unboundedness in *F(x)* may be

manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against r_1 can be made.

Unboundedness is x is best avoided by placing finite upper and lower bounds on the variables. see also the *Minor damping parameter*.

(*Default = 1.0E+20*)

Unbounded step size r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If α exceeds r, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`.

If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against r_1 can be made.

Unboundedness is x is best avoided by placing finite upper and lower bounds on the variables. see also the *Minor damping parameter*.

(*Default = 1.0E+10*)

Verify option i

This option refers to a finite-difference check on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as "correct".

Gradient verification occurs before the problem is scaled, and before the first basis is factorized. (Hence, it occurs before the basic variables are set to satisfy the general constraints $Ax + s = 0$.)

(*Default = 0*)

0 Only a "cheap" test is performed, requiring three evaluations of the nonlinear objective (if any) and two evaluations of the nonlinear constraints. *Verify No* is an equivalent option.

1 A more reliable check is made on each component of the objective gradient. *Verify objective gradients* is an equivalent option.

2 A check is made on each column of the Jacobian matrix

	associated with the nonlinear constraints. *Verify constraint gradients* is an equivalent option.
3	A detailed check is made on both the objective and the Jacobian. *Verify*, *Verify gradients*, and *Verify Yes* are equivalent options.
-1	No checking is performed.
Weight on linear objective r	The keywords invokes the so-called *composite objective* technique, if the first solution obtained infeasible, and if the objective function contains linear terms. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear objective. At each infeasible iteration, the objective function is defined to be $minimize_x \ \sigma w(c^T x) + (sum\ of\ infeasibilities)$ where $\sigma = 1$ for minimization and $\sigma = -1$ for maximization and c is the linear objective. If an "optimal" solution is reached while still infeasible, *w* is reduced by a factor of 10. This helps to allow for the possibility that the initial *w* is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending towards zero. The effect of *w* is disabled after five such reductions, or if a feasible solution is obtained. This option is intended mainly for linear programs. It is unlikely to helpful if the objective function is nonlinear. (*Default = 0.0*)

9. Acknowledgements

This chapter is a minor revision to the earlier description of the GAMS/MINOS system found in the GAMS User's Guide. Phillip Gill, Walter Murray, Bruce A. Murtagh, Michael A. Saunders, and Margaret H. Wright were the authors of this earlier description. The section on the GAMS/MINOS log file was initially put together by Michael Saunders and Arne Drud.

10. References

Bartels, R.H. (1971), A stabilization of the simplex method, Numerische Mathematik 16, 414-434.

Bartels, R.H. and G. H. Golub (1969), The simplex method of linear programming using the LU decomposition, *Communications of the ACM* 12, 266-268.

Dantzig, G.B. (1963), *Linear Programming and Extensions,* Princeton University Press, Princeton, New Jersey.

Davidon, W.C. (1959), Variable metric methods for minimization, A.E.C. Res. and Develop. Report ANL-5990, Argonne National Laboratory, Argonne, IL.

Fourer, R. (1982), Solving staircase linear programs by the simplex method, *Mathematical Programming* 23, 274-313.

Gill, P.E. , W, Murray, M. A. Saunders and M. H. Wright (1979), Two step-length algorithms for numerical optimization, Report SOL 79-25, Department of Operations Research, Stanford University, Stanford, California

Gill, P.E. , W, Murray, M. A. Saunders and M. H. Wright (1987), Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications* 88/89, 239-270.

Murtagh, B.A. and M. A. Saunders (1978), Large-scale linearly constrained optimization, *Mathematical Programming* 14, 41-72.

Murtagh, B.A. and M. A. Saunders (1982), A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Mathematic Programming Study* 16, *Algorithm for Constrained Minimization of Smooth Nonlinear Function,* 84-117.

Murtagh, B.A. and M. A. Saunders (1983), MINOS 5.0 User‘s Guide, Report SOL 83-20, Department of Operations Research, Stanford University, (Revised as MINOS 5.1 User‘s Guide, Report SOL 83-20R, 1987.)

Reid, J.K. (1976), Fortran subroutines for handling sparse linear programming bases, Report R8269, Atomic Energy Research Establishment, Harwell, England.

Reid, J.K. (1982), A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases, *Mathematical Programming* 24, 55-69.

Robinson, S,M. (1972), A quadratically convergent algorithm for general nonlinear programming problems, *Mathematical Programming* 3, 145-156.

Wolfe, P (1962), The reduced-gradient method, unpublished manuscript, RAND Corporation.