MINOS

Contents

Articles

References

MINOS

Introduction

TOMLAB /MINOS (hereafter referred to as MINOS) is a linear and nonlinear programming system, designed to solve large-scale constrained optimization problems of the following form:

$$\begin{array}{lll} \min\limits_{x,y} & F(x) + c^T x + d^T y & \\ s/t \quad b_1 & \leq f(x) + A_1 y \leq & b_2, \\ b_3 & \leq A_2 x + A_3 y \leq & b_4, \\ l & \leq (x, y) \leq & u, \end{array}$$

where the vectors b_i, c, d, l, u and the matrices A_i are constant, $F(x)$ is a nonlinear function of some of the variables, and $f(x)$ is a vector of nonlinear functions. The nonlinearities (if present) may be of a general nature but must be smooth and preferably "almost linear", in the sense that they should not change radically with small changes in the variables. We make the following definitions:

x	the nonlinear variables
y	the linear variables
(x, y)	the vector $(x\ y)^T$
(1.1)	the objective function
(1.2)	the nonlinear constraints
(1.3)	the linear constraints
(1.4)	the bounds on the variables
m	the total number of general constraints in (2) and (3)
n	the total number of variables in x and y
m_1	the number of nonlinear constraints (the dimension of $f(x)$)
n_1	the number of nonlinear variables (the dimension of x)
n'_1	the number of nonlinear objective variables (in $F(x)$)
n''_1	the number of nonlinear Jacobian variables (in $f(x)$)

A large-scale problem is one in which m and n are several hundred or several thousand. MINOS takes advantage of the fact that the constraint matrices Ai and the partial derivatives $\partial f_i(x)/\partial x_j$ are typically sparse (contain many zeros).

The dimensions n'_1 and n''_1 allow for the fact that $F(x)$ and $f(x)$ may involve different sets of nonlinear variables "x". The two sets of variables *always overlap*, in the sense that the shorter "x" is always the same as the beginning of the other. If x is the same in both cases, we have $n_1 = n'_1 = n''_1$. Otherwise, we define the number of nonlinear variables to be $n_1 = \max(n'_1, n''_1)$.

In the following sections we introduce more terminology and give an overview of the MINOS optimization algorithms and the main system features.

Linear Programming

When $F(x)$ 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. We also convert all general constraints into equalities with the aid of slack variables s, so that the only inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\min_{x,\ s} \quad c^T x \text{subject to} Ax + s = b, \quad l \leq (x,\ s) \leq u.$$

When the constraints are linear, the bounds on the slacks are defined so that $b = 0$. When there are nonlinear constraints, some elements of b are nonzero.

In the mathematical programming world, x and s are sometimes called *structural* variables and *logical* variables. Their upper and lower bounds are fundamental to problem formulations and solution algorithms. Some of the components of l may be $-\infty$ and those of u may be $+\infty$. If $l_j = u_j$, a variable is said to be *fixed*, and if its bounds are $-\infty$ and $+\infty$, the variable is called *free*.

Within MINOS, a point (x, s) is said to be *feasible* if the following are true:

- The constraints $Ax + s = b$ are satisfied to within machine precision $\approx 10^{-15}$.
- The bounds $l <= (x, s) <= u$ are satisfied to within a *feasibility tolerance* δ_{fea} □ 10^{-6}.
- The nonlinear constraints (32) are satisfied to within a *row tolerance* δ_{row} □ 10^{-6}.

Tolerances such as δ_{fea} and δ_{row} may be specified by setting Feasibility tolerance and Row tolerance.

MINOS solves linear programs using a reliable implementation of the *primal simplex method, in which the constraints* Ax + s = b *are partitioned into the form*

$$Bx_B + Nx_N = b,$$

where the *basis matrix B* is a square and nonsingular submatrix of (*A I*). The elements of x_B and x_N are called the basic and nonbasic variables respectively. Together, they are a permutation of the vector (x, s). Certain *dual variables* π *and* reduced costs d_N *are defined by the equations*

$$B^T\pi = c_B, \qquad d_N = c_N - N^T\pi,$$

where (c_B, c_N) is a permutation of the objective vector $(c, 0)$.

At a feasible point, nonbasic variables are typically equal to one of their bounds, and basic variables are somewhere between their bounds. To reach an optimal solution, the simplex method performs a sequence of *iterations* of the following general nature. With guidance from d_N, a nonbasic variable is chosen to move from its current value, and the basic variables are adjusted to satisfy the constraints. Usually one of the basic variables reaches a bound. The basis partition is then redefined with a column of B being replaced by a column of N. When no such interchange can be found to reduce the value of $c^T x$, the current solution is optimal.

The simplex method

For convenience, let x denote the variables (x, s). The main steps in a simplex iteration are as follows:

Compute dual variables: Solve $B^T\pi = c_B$.

Price: Compute some or all of the reduced costs $d_N = c_N - N^T\pi$ to determine if a favorable nonbasic column a_q exists.

Compute search direction: Solve $Bp_B = \pm a_q$ to determine the basic components of a search direction p along which the objective is improved. (The nonbasic elements of p are $p_N = 0$, except for ±1 for the element corresponding to a_q.)

Find maximum steplength: Find the largest steplength α_{max} such that $x + \alpha_{max} p$ continues to satisfy the bounds on the variables. The steplength may be determined by the new nonbasic variable reaching its opposite bound, but normally some basic variable will reach a bound first.

Update: Take the step $\alpha_{\max}$. If this was determined by a basic variable, interchange the corresponding column of B with column a_q from N.

When a starting basis is chosen and the basic variables x_B are first computed, 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. This is similar to the subsequent "Phase 2" procedure just described.

The feasibility tolerance δ_{fea} is used to determine which Phase is in effect. A similar *optimality tolerance* δ_{opt} is used during pricing to judge whether any reduced costs are significantly large. (This tolerance is scaled by $||\pi||$, a measure of the size of the current π.)

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: $l_j - \delta_{fea} \leq x_j \leq u_j + \delta_{fea}$. In rare cases, even a few nonbasic variables might lie outside their bounds by as much as δ_{fea}.

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. The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

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. MINOS solves such problems using a *reduced-gradient* method combined with a *quasi-Newton* method that generally leads to superlinear convergence. The implementation follows that described in Murtagh and Saunders.

As a slight generalization of the constraints $Ax + s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where x_S is a set of *superbasic variables*. As before, the nonbasic variables are normally equal to one of their bounds, while the basic *and* superbasic variables lie somewhere between their bounds (to within δ_{fea}). Let the number of superbasic variables be nS, the number of columns in S. At a solution, n_S will be no more than n_1, the number of nonlinear variables, and it is often much smaller than this. In many real-life cases we have found that n_S remains reasonably small, say 200 or less, regardless of the size of the problem. This is one reason why MINOS has proved to be a practical tool.

In the reduced-gradient method, x_S is regarded as a set of "independent variables" that are allowed to move in any desirable direction to reduce the objective function (or the sum of infeasibilities). The basic variables are then 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, one of the nonbasic variables is selected to be added to S, and the process is repeated with an increased value of n_S. At all stages, if a basic or superbasic variable encounters one of its bounds, that variable is made nonbasic and the value of n_S is reduced by one.

For linear programs, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variables oscillating between 0 and 1. (In general, a step of the simplex method *or* the reduced-gradient method is called a *minor iteration*.)

A certain matrix Z is needed 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* factors of the basis matrix *B*, it is possible to compute products of the form Z_q and $Z^T g$ 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. (In the description below, the reduced-gradient vector satisfies $d_S = Z^T g$, and the search direction satisfies $p = Zp_S$.)

An important part of MINOS is the quasi-Newton method used to optimize the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the *B*, *S*, *N* partition remains constant. It requires a dense upper-triangular matrix *R* of dimension n_S, which is updated in various ways to approximate the *reduced Hessian* :

$$R^T R \approx Z^T H Z,$$

where *H* is the *Hessian* of the objective function, i.e. the matrix of second derivatives of *F* (*x*). As for unconstrained optimization, the storage required for *R* is sometimes a limiting factor.

The reduced-gradient method

Let *g* be the gradient of the nonlinear objective. The main steps in a reduced-gradient iteration are as follows:

Compute dual variables and reduced gradient: Solve $B^T\pi = g_B$ and compute the reduced-gradient vector $d_S = g_S - S^T\pi$.

Price: If $||d_S||$ is sufficiently small, compute some or all of the reduced costs $d_N = g_N - N^T\pi$ to determine if a favorable nonbasic column a_q exists. If so, move that column from *N* into *S*, expanding *R* accordingly.

Compute search direction: Solve $R^T R p_S = -d_S$ and $Bp_B = -Sp_S$ to determine the superbasic and basic components of a search direction *p* along which the objective is improved. (The nonbasic elements of *p* are $p_N = 0$.)

Find maximum steplength: Find the largest steplength $\alpha_{\max}$ such that $x + \alpha_{\max} p$ continues to satisfy the bounds on the variables.

Perform linesearch: Find an approximate solution to the one-dimensional problem

$$\min_{\alpha} \quad F(x + \alpha p) \text{subject to} 0 \leq \alpha \leq \alpha_{max}.$$

Update (quasi-Newton): Take the step α. Apply a quasi-Newton update to *R* to account for this step.

Update (basis change): If a superbasic variable reached a bound, move it from *S* into *N* . If a basic variable reached a bound, find a suitable superbasic variable to move from *S* into *B*, and move the basic variable into *N* . Update *R* if necessary.

At an optimum, the reduced gradient d_S should be zero. MINOS terminates when $||d_S|| <= \delta_{opt}||\pi||$ and the reduced costs (component of *dN*) are all sufficiently positive or negative, as judged by the same quantity $delta_{\text{opt}}||\pi||$.

In the linesearch, *F* (*x* + αp) really means the objective function evaluated at the point (*x, y, s*) + αp. This steplength procedure is another important part of MINOS. Two different procedures are used, depending on whether or not all gradients are known analytically. The number of nonlinear function evaluations required may be influenced by setting the Linesearch tolerance in the SPECS file.

Normally, the objective function *F* (*x*) will never be evaluated at a point *x* unless that point satisfies the linear constraints and the bounds on the variables. An exception is during a finite-difference check on the calculation of gradient elements. This check is performed at the *starting point* x_0 , which takes default values or may be specified. MINOS ensures that the bounds $l <= x_0 <= u$ are satisfied, but in general the starting point will not satisfy the general linear constraints. If *F* (x_0) is undefined, the gradient check should be suppressed (Verify level -1), or the starting point should be redefined.

Problems with Nonlinear Constraints

If any of the constraints are nonlinear, MINOS employs a *projected Lagrangian* algorithm, based on a method due to Robinson. 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, y_k) be an estimate of the variables, and let γ_k be an estimate of the Lagrange multipliers (dual variables) associated with the nonlinear constraints. The constraints are linearized by changing $f(x)$ to its linear approximation:

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

or more briefly $\bar{f} = f_k + J_k(x - x_k)$, where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k. (The *i*-th row of the Jacobian is the gradient vector for the *i*-th nonlinear constraint function.) The subproblem to be solved during the *k*-th major iteration is then

$$\begin{array}{lccc} \min_{x,y} & F(x) + c^T x + d^T y - \lambda_k^T f_d + \frac{1}{2}\rho_k \|f_d\|^2 & & \\ s/t & b_1 \leq \bar{f} + A_1 y \leq & & b_2 \\ & b_3 \leq A_2 x + A_3 y \leq & & b_4, \\ & l \leq (x, y) \leq & & u, \end{array}$$

where $f_d = f - \bar{f}$ is the difference between $f(x)$ and its linearization. The objective function is called an *augmented Lagrangian*. The scalar ρ_k is a *penalty parameter*, and the term involving ρ_k is a modified *quadratic penalty function*. MINOS uses the reduced-gradient method to solve each subproblem. As before, slack variables are introduced and the vectors b_i are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} S_1 \\ S_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}.$$

We refer to this system as $Ax + s = b$ as in the linear case. The Jacobian *Jk* is treated as a sparse matrix, the same as the matrices *A*1 , *A*2 and *A*3 . The quantities J_k , b, λ_k and ϱ_k change each major iteration.

The projected Lagrangian method

For convenience, suppose that all variables and constraints are nonlinear. The main steps in a major iteration are as follows:

Solve subproblem: Find an approximate solution $(\bar{x}, \bar{\lambda})$ to the *k*th subproblem.

Compute search direction: Adjust the elements of $\bar{\lambda}$ if necessary (if they have the wrong sign).

Define a search direction $(\Delta x,\ \Delta\lambda) = (\bar{x} - x_k, \bar{\lambda} - \lambda_k)$.

Find steplength: Choose a steplength σ such that some merit function $M(x, \lambda)$ has a suitable value at the point $(x_k + \sigma\Delta x,\ \lambda_k + \sigma\Delta\lambda)$.

Update: Take the step σ to obtain $(x_{k+1},\ \lambda_{k+1})$. In some cases, adjust ρ_k.

For the first major iteration, the nonlinear constraints are ignored and minor iterations are performed until the original linear constraints are satisfied.

The initial Lagrange multiplier estimate is typically $\lambda_k = 0$ (though it can be provided by the user). If a subproblem terminates early, some elements of the new estimate $\bar{\lambda}$ may be changed to zero.

The penalty parameter initially takes a certain default value $\rho_k = 100.0/m_1$, where m_1 is the number of nonlinear constraints. (A value *r* times as big is obtained by specifying Penalty parameter *r*.) For later major iterations, ρ_k is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many cases it is safe to specify Penalty parameter 0.0 at the beginning, particularly if a problem is only mildly nonlinear. This may improve the overall efficiency.

In the output from 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 iterations.

For certain rare classes of problem it is safe to request the values $\lambda_k = 0$ and $\rho_k = 0$ for all subproblems by specifying Lagrangian = No (in which case the nonlinear constraint functions are evaluated only once per major iteration). However for general problems, convergence is much more likely with the default setting, Lagrangian = Yes.

The merit function

Unfortunately, it is not known how to define a merit function $M(x, \lambda)$ that can be *reduced* at every major iteration. As a result, there is no guarantee that the projected Lagrangian method described above will converge from an arbitrary starting point. This has been the principal theoretical gap in MINOS, finally resolved by the PhD research of Michael Friedlander. The main features needed to stabilize MINOS are:

- To relax the linearized constraints via an ℓ_1 penalty function.
- To repeat a major iteration with increased ρ_k (and more relaxed linearized constraints) if the nonlinear constraint violation would increase too much.

In practice, the method of MINOS 5.51 often does converge and a good *rate* of convergence is often achieved in the final major iterations, particularly if the constraint functions are "nearly linear". As a precaution, MINOS prevents radical changes from one major iteration to the next. Where possible, the steplength is chosen to be $\sigma = 1$, so that each new estimate of the solution is $(x_{k+1},\ \lambda_{k+1}) = (\bar{x}, \bar{\lambda})$, the solution of the subproblem. If this point is "too different", a shorter steplength $\sigma < 1$ is chosen.

If the major iterations for a particular model do not appear to be converging, some of the following actions may help:

1. Specify initial activity levels for the nonlinear variables as carefully as possible.
2. Include sensible upper and lower bounds on all variables.
3. Specify a Major damping parameter that is lower than the default value. This tends to make *s* smaller.
4. Specify a Penalty parameter that is higher than the default value. This tends to prevent excessive departures from the constraint linearization.

Problem Formulation

In general, it is worthwhile expending considerable prior analysis to make the constraints completely linear if at all possible. Sometimes a simple transformation will suffice. For example, a pipeline optimization problem has pressure drop constraints of the form

$$\frac{K_1}{d_1^{4.814}} + \frac{K_2}{d_2^{4.814}} + \cdots \leq P_T^2 - P_0^2,$$

where d_i are the design variables (pipe diameters) and the other terms are constant. These constraints are highly nonlinear, but by redefining the decision variables to be $x_i = 1/d_i^{4.814}$ we can make the constraints linear. Even if the objective function becomes more nonlinear by such a transformation (and this usually happens), the advantages of having linear constraints greatly outweigh this.

Similarly, it is usually best not to move nonlinearities from the objective function into the constraints. For example, we should *not* replace "$\min\ F(x)$" by

$$\min\quad z\quad \text{subject to}\quad F(x) - z = 0.$$

Scaling is a very important matter during problem formulation. A general rule is to scale both the data and the variables to be as close to 1.0 as possible. In general we suggest the range 1.0 to 10.0. When conflicts arise, one should sacrifice the objective function in favor of the constraints. Real-world problems tend to have a natural scaling within each constraint, as long as the variables are expressed in consistent physical units. Hence it is often sufficient to apply a scale factor to each row. MINOS has options to scale the rows and columns of the constraint matrix automatically. By default, only the linear rows and columns are scaled, and the procedure is reliable. If you request that the nonlinear constraints and variables be scaled, bear in mind that the scale factors are determined by the initial Jacobian $J(x_0)$, which may differ considerably from $J(x)$ at a solution.

Finally, *upper and lower bounds* on the variables (and on the constraints) are extremely useful for confining the region over which optimization has to be performed. If sensible values are known, they should always be used. They are also important for avoiding singularities in the nonlinear functions. Note that bounds may be violated slightly by as much as the feasibility tolerance δ_{fea}. Hence, if $\sqrt{x_2}$ or $\log x_2$ appear (for example) and if $\delta_{\text{fea}} = 10^{-6}$, the lower bound on x_2 would normally have to be at least 10^{-5}. If it is *known* that x_2 will be at least 0.5 (say) at a solution, then its lower bound should be 0.5.

For a detailed discussion of many aspects of numerical optimization, see Gill, Murray and Wright; in particular, see Chapter 8 for much invaluable advice on problem formulation and assessment of results.

Restrictions

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), especially near the desired solution. The functions need not be separable.

A certain "feasible" region is defined by the general constraints and the bounds on the variables. If the objective is convex within this region and if the feasible region itself is convex, 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 starting 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.

Integer restrictions cannot be imposed directly. If a variable x_j is required to be 0 or 1, a common ploy is to include a quadratic term $x_j(1 - x_j)$ in the objective function. MINOS will indeed terminate with $x_j = 0$ or 1, but inevitably the final solution will just be a local optimum. (Note that the quadratic is negative definite. MINOS will find a global minimum for quadratic functions that are positive definite or positive semidefinite, assuming the constraints are linear.)

Solver Options

- MINOS Solver Options

File Output

- MINOS File Output

MINOS Solver Options

Options for All Problems

The following options have the same purpose for all problems, whether they linear or nonlinear.

```
Check frequency  k  Default = 60
```

Every *k*-th minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution *x* satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = b$, where *s* is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax - s$ is computed. If the largest component of *r* is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Check frequency 1 is useful for debugging purposes, but otherwise this option should not be needed.

```
Cycle limit      l  Default = 1
Cycle print      p  Default = 1
Cycle tolerance  t  Default = 0.0
Phantom columns  c  Default = 0
Phantom elements e  Default = 0
```

```
Debug level  l  Default = 0
```

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

$l = 0$	No debug output.
$l = 2$	(or more) Output from m5setx showing the maximum residual after a row check.
$l = 40$	Output from lu8rpc (which updates the *LU* factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
$l = 50$	Output from lu1mar (which computes the *LU* factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
$l = 100$	Output from m2bfac and m5log listing the basic and superbasic variables and their values at every iteration.

```
Expand frequency  k  Default = 10000
```

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

"Cycling" can occur only if zero steplengths are allowed. Here, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the Feasibility tolerance is *d*. Over a period of *k* iterations, the tolerance actually used by MINOS increases from 0.5δ to δ (in steps of $0.5\delta/k$).

Every *k* iterations, or when feasibility and optimality are first detected, a resetting procedure eliminates any infeasible nonbasic variables. Some additional iterations may be needed to restore feasibility and optimality. Increasing *k* reduces that likelihood, but it gives less freedom to choose large pivot elements during basis changes. (See Pivot tolerance.)

```
Factorization frequency k Default = 100 (LP) or 50 (NLP)
```

With linear programs, most iterations cause a basis change, in which one column of the basis matrix *B* is replaced by another. The *LU* factors of *B* must be updated accordingly. At most *k* updates are performed before the current *B* is factorized directly.

Each update tends to add nonzeros to the *LU* factors. Since the updating method is stable, *k* mainly affects the efficiency of minor iterations, rather than stability.

High values of *k* (such as 100 or 200) may be more efficient on "dense" problems, when the factors of *B* tend to have two or three times as many nonzeros as *B* itself. Lower values of *k* may be more efficient on problems that are very sparse.

```
Feasibility tolerance t Default = 1.0e-6
```

This sets the feasibility tolerance $\delta_{\text{fea}} = t$ (see #Options for Linear Programming). A variable or constraint is considered *feasible* if it does not lie outside its bounds by more than δ_{fea}.

MINOS first attempts to satisfy the linear constraints and bounds. 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 *t* 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. MINOS does not attempt to find a solution that minimizes the sum.

For Scale option 1 or 2, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful). The final unscaled solution can therefore be infeasible by an unpredictable amount, depending on the size of the scale factors. Precautions are taken so that in a "feasible solution" the original variables will never be infeasible by more than 0.1. Values that large are very unlikely.

```
Iterations limit k Default = 3m
```

MINOS stops after *k* iterations even if the simplex method has not yet reached a solution. If *k* = 0, no iterations are performed, but the starting point is tested for both feasibility and optimality.

```
LU factor tolerance t1 Default = 100.0 (LP) or 5.0 (NLP)
```

```
LU update tolerance t2 Default = 10.0 (LP) or 5.0 (NLP)
```

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $t_1, t_2 >= 1.0$. The matrix *L* is a product of matrices of the form

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

where the multipliers μ satisfy $|\mu| = t_i$. Values near 1.0 favor stability, while larger values favor sparsity. The default values usually strike a good compromise. For large and relatively dense problems, t_1 = 10.0 or 5.0 (say) may give a useful improvement in stability without impairing sparsity to a serious degree.

For certain very regular structures (e.g., band matrices) it may be necessary to reduce t_1 and/or t_2 in order to achieve stability. For example, if the columns of *A* include a submatrix of the form

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

one should set both t_1 and t_2 to values in the range $1.0 \leq t_i < 4.0$

LU density tolerance t_3 Default = 0.5

LU singularity tolerance t_4 Default = $E^{0.67} \approx 3.25e{-}11$

The density tolerance t_3 is used during *LU* factorization of the basis matrix. Columns of *L* and rows of *U* are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds t_3, the Markowitz strategy for choosing pivots is altered to reduce the time spent searching for each remaining pivot. Raising the density tolerance towards 1.0 may give slightly sparser *LU* factors, with a slight increase in factorization time.

The singularity tolerance t_4 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of *U* are tested as follows: if $|U_{jj}| \leq t_4$ or $|U_{jj}| < t_4 \max_i |U_{ij}|$, 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 exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting t_4 = 1.0e-5, say, may help cause a judicious change of basis.

Maximize Default

Minimize Default

This specifies the required direction of optimization.

Multiple price k Default = 1

It is not normal to set *k* > 1 for linear programs, as it causes MINOS to use the reduced-gradient method rather than the simplex method. The number of iterations, and the total work, are likely to increase.

The reduced-gradient iterations do *not* correspond to the very efficient multiple pricing "minor iterations" carried out by certain commercial linear programming systems. Such systems require storage for *k* dense vectors of dimension *m*, so that *k* is usually limited to 5 or 6. In MINOS, the total storage requirements increase only slightly with *k*. (The Superbasics limit must be at least *k*.)

Optimality tolerance t Default = 1.0e-6

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 is declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy $d_j/||\pi|| \geq -t$ or $d_j/||\pi|| \leq t$ respectively, and if $||d_j||/||\pi|| \leq t$ for superbasic variables.

In those tests, $||\pi||$ is a measure of the size of the dual variables. It is included to make the tests independent of a large scale factor on the objective function. The quantity actually used is defined by

$$\sigma = \sum_{i=1}^{m} |\pi_i|, ||\pi|| = \max\{\sigma/\sqrt{m}, 1.0\},$$

so that only scale factors larger than 1.0 are allowed for. If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against *t*.

```
Partial price p Default = 10 (LP) or 1 (NLP)
```

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 $p = 1$, all columns of the constraint matrix (A I) are searched. Otherwise, A and I are partitioned to give p roughly equal segments A_j , I_j ($j = 1$ to p). If the previous pricing search was successful on A_j , I_j, the next search begins on the segments A_j+1 , I_j+1 . (Subscripts are modulo p.)

If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (Several may be selected if multiple pricing has been specified.) If nothing is found, the search continues on the next segments A_j+2 , I_j+2 , and so on.

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

```
Pivot tolerance t  Default = E^2/3 ≈ 10^-11
```

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular. When x changes to $x + \alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.

For linear problems, elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance *t*. For nonlinear problems, elements smaller than $t||p||$ are ignored.

It is common for two or more variables to reach a bound at essentially the same time. In such cases, the Feasibility tolerance provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small Feasibility tolerances should therefore not be specified.

To a lesser extent, the Expand frequency also provides some freedom to maximize the pivot element. Excessively *large* Expand frequencies should therefore not be specified.

```
Scale option              l   Default = 2 (LP) or 1 (NLP)
Scale                     Yes
Scale                     No
Scale linear variables        Same as Scale option 1
Scale nonlinear variables     Same as Scale option 2
Scale all variables           Same as Scale option 2
Scale tolerance           t   Default = 0.9
Scale, Print
Scale, Print, Tolerance   t
```

Three scale options are available as follows:

l = 0	No scaling. If storage is at a premium, this option saves *m* + *n* words of workspace.
l = 1	possible to 1.0. This sometimes improves the performance of the solution procedures.
l = 2	helpful if the right-hand side *b* or the solution *x* is large. This takes into account columns of (*A I*) that are fixed or have positive lower bounds or negative upper bounds.

Scale Yes sets the default scaling. (*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 tried at first. Scale option 2 gives scales that depend on the initial Jacobian, and should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

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

All forms except Scale option may specify a tolerance *t*, where $0 < t < 1$ (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 ratio of the largest and smallest nonzero coefficients in each column:

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

If $\max_j \rho_j$ is less than *t* times its previous value, another scaling pass is performed to adjust the row and column scales. Raising *t* 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, Print or Scale tolerance both set the default scaling.

```
Weight on linear objective w Default = 0.0
```

This keyword invokes the so-called *composite objective* technique, if the first solution obtained is 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

$$\min_x \quad \sigma w(c^T x) + (\text{sum of infeasibilities}),$$

where σ = 1 for minimization, σ = -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 that the sum of infeasibilities is tending towards zero.

The effect of *w* is disabled after 5 such reductions, or if a feasible solution is obtained.

The Weight option is intended mainly for linear programs. It is unlikely to be helpful on nonlinear problems.

MINOS File Output

The PRINT file

The following information is output to the PRINT file during the solution process. The longest line of output is 124 characters.

- A listing of the SPECS file, if any.
- The selected options.
- An estimate of the storage needed and the amount available.
- Some statistics about the problem data.
- The storage available for the LU factors of the basis matrix.
- A log from the scaling procedure, if Scaleoption > 0.
- Notes about the initial basis obtained from CRASH or a BASIS file.
- The major iteration log.
- The minor iteration log.
- Basis factorization statistics.
- The EXIT condition and some statistics about the solution obtained.
- The printed solution, if requested.

The last five items are described in the following sections.

The major iteration log

Problems with nonlinear constraints require several *major iterations* to reach a solution, each involving the so- lution of an *LC subproblem* (a linearly constrained subproblem that generates search directions for x and λ). If Printlevel = 0, one line of information is output to the PRINT file each major iteration. An example log is shown in <xr id="fig:exampleLog" />.

<figure id="fig:exampleLog">

```
Major minor   total ninf step     objective     Feasible Optimal  nsb   ncon     LU penalty BSwap
     1     1T      1    0 0.0E+00  0.00000000E+00 0.0E+00 1.2E+01    8      4     31 1.0E-01     0
     2    13      14    0 1.0E+00  2.67011596E+00 4.4E-06 2.8E-03    7     23     56 1.0E-01     8
 Completion Full    now requested
     3     3      17    0 1.0E+00  2.67009870E+00 3.1E-08 1.4E-06    7     29     41 1.0E-01     0
     4     0      17    0 1.0E+00  2.67009863E+00 5.6E-17 1.4E-06    7     30     41 1.0E-02     0
```

<caption>The Major Iteration log </figure>

Label	Description
Major	The current major iteration number.
minor	is the number of iterations required by both the feasibility and optimality phases of the QP sub- problem. Generally, Mnr will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution.
total	The total number of minor iterations.
ninf	The number of infeasibilities in the LC subproblem. Normally 0, because the bounds on the linearized constraints are relaxed in several stages until the constraints are "feasible".
step	The step length a taken along the current search direction p. The variables x have just been changed to $x + ap$. On reasonably well-behaved problems, step = 1.0 as the solution is approached, meaning the new estimate of $(x, ?)$ is the solution of the LC subproblem.
objective	The value of true objective function.

Feasible	The value of rowerr, the maximum component of the scaled nonlinear constraint residual. The solution is regarded as acceptably feasible if Feasbl is less than the Row tolerance.
Optimal	The value of maxgap, the maximum complementarity gap. It is an estimate of the degree of nonop- timality of the reduced costs. Both Feasible and Optimal are small in the neighborhood of a solution.
nsb	The current number of superbasic variables.
ncon	The number of times subroutine funcon has been called to evaluate the nonlinear constraint functions. The Jacobian has been evaluated or approximated essentially the same number of times. (Function evaluations needed to estimate the Jacobian by finite differences are not included.)
LU	The number of nonzeros in the sparse LU factors of the basis matrix on completion of the LC subproblem. (The factors are computed at the start of each major iteration, and updated during minor iterations whenever a basis change occurs.) As the solution is approached and the minor iterations decrease towards zero, LU reflects the number of nonzeros in the LU factors at the start of the LC subproblem.
penalty	The penalty parameter ϱ_k used in the modified augmented Lagrangian that defines the objective function for the LC subproblem.
BSwap	The number of columns of the basis matrix B that were swapped with columns of S to improve the condition of B. The swaps are determined by an LU factorization of the rectangular matrix $BS = (B\ S)^T$ with stability being favored more than sparsity.

<figure id="fig:minorIterLog">

```
    Itn ph pp     rg    +sbs  -sbs   -bs  step    pivot   ninf  sinf,objective     L     U ncp  nobj  ncon  nsb Hmod cond(H) conv

      1  1  1 -1.0E+00     2     2     1 3.0E+01  1.0E+00    1  1.35000000E+02     0    19   0

      2  1  1 -1.0E+00    27    27   102 7.0E+01  1.0E+00    1  1.05000000E+02     0    19   0

      3  1  1 -1.0E+00     3     3    27 3.0E+01 -1.0E+00    1  3.50000000E+01     1    19   0

      4  1  1 -1.0E+00    28    28    26 4.9E-11  1.0E+00    1  5.00000000E+00     1    20   0

      5  1  1 -1.0E+00    47    47     2 4.9E-11  1.0E+00    1  5.00000000E+00     1    20   0

      6  1  1  1.0E+00    27    27   101 5.0E+00 -1.0E+00    1  5.00000000E+00     2    20   0


 Itn      6 -- feasible solution.  Objective =  -1.818044887E+02


      7  3  1 -1.7E+01    87     0     0 1.0E+00  0.0E+00    0 -2.77020571E+02     4    21   0     6     0    1  1 0 1.0E+00 FFTT

      8  3  1 -1.7E+01    72     0     0 1.9E-01  0.0E+00    0 -3.05336895E+02     4    21   0     8     0    2  1 0 5.5E+00 FFTT

      9  3  1 -2.3E+01    41     0     0 1.0E+00  0.0E+00    0 -4.43743832E+02     4    21   0     9     0    3  1 0 6.5E+00 FFFF

     10  4  1  6.6E-01     0     0     0 6.0E+00  0.0E+00    0 -5.64075338E+02     4    21   0    11     0    3  1 0 3.5E+00 FFTT
...


    Itn ph pp     rg    +sbs  -sbs   -bs  step    pivot   ninf  sinf,objective     L     U ncp  nobj  ncon  nsb Hmod cond(H) conv

    161  4  1  8.8E-03     0    73    71 4.2E+00  1.0E+00    0 -1.73532497E+03     4    20   0   340     0   17  1 1 9.6E+00 TTTF

    162  3  1 -3.5E-02     6     0     0 1.5E+00  0.0E+00    0 -1.73533264E+03     4    20   0   342     0   18  1 0 1.3E+02 TTFF

    163  4  1  2.9E-02     0     0     0 4.5E+00  0.0E+00    0 -1.73533617E+03     4    20   0   344     0   18  1 0 2.0E+01 TTFF

    164  4  1  2.1E-02     0     0     0 2.3E+01  0.0E+00    0 -1.73538331E+03     4    20   0   347     0   18  1 0 9.8E+00 TTFF

    165  4  1  3.0E-02     0     0     0 5.0E+00  0.0E+00    0 -1.73552261E+03     4    20   0   349     0   18  1 0 2.1E+01 TTFF

    166  4  1  1.2E-02     0     0     0 1.0E+00  0.0E+00    0 -1.73556089E+03     4    20   0   350     0   18  1 0 2.2E+01 TTTF

 tolrg  reduced to  1.162E-03      lvltol = 1

    167  4  1  2.3E-03     0     0     0 1.0E+00  0.0E+00    0 -1.73556922E+03     4    20   0   351     0   18  1 0 2.2E+01 TTFF

    168  4  1  1.2E-03     0     0     0 7.9E-01  0.0E+00    0 -1.73556953E+03     4    20   0   353     0   18  1 0 2.1E+01 TTFF

    169  4  1  1.0E-04     0     0     0 1.0E+00  0.0E+00    0 -1.73556958E+03     4    20   0   354     0   18  1 0 2.0E+01 TTTT

 tolrg  reduced to  1.013E-05      lvltol = 1

    170  4  1  2.9E-05     0     0     0 1.1E+00  0.0E+00    0 -1.73556958E+03     4    20   0   356     0   18  1 0 1.7E+01 TTFF

    171  4  1  1.0E-05     0     0     0 1.0E+00  0.0E+00    0 -1.73556958E+03     4    20   0   357     0   18  1 0 1.7E+01 TTFF
```

```
   172  4  1  1.5E-06     0     0     0 1.2E+00  0.0E+00    0 -1.73556958E+03     4    20   0   359     0   18  1 0 1.7E+01 TTTF
tolrg  reduced to  1.000E-06      lvltol = 2
   173  4  1  2.4E-07     0     0     0 1.0E+00  0.0E+00    0 -1.73556958E+03     4    20   0   360     0   18  1 0 1.7E+01 TTTF


Biggest dj =  3.583E-03 (variable     25)   norm rg =  2.402E-07   norm pi =  1.000E+00
```

<caption>The Minor Iteration log </figure>

The minor iteration log

If Printlevel = 1, one line of information is output to the PRINT file every *k*th minor iteration, where *k* is the specified Print frequency (default *k* = 100). A heading is printed periodically. Problem t5weapon gives the log shown in <xr id="fig:minorIterLog" />.

Label	Description
Itn	The current minor iteration number.
ph	The current phase of the solution procedure: 1 Phase 1 simplex method, trying to satisfy the linear constraints. The current solution is an infeasible vertex. 2 Phase 2 simplex method, solving a linear program. 3 Reduced-gradient method. A nonbasic variable has just become superbasic. 4 Reduced-gradient method, optimizing the current set of superbasic variables.
pp	The Partial Price indicator. The variable selected by the last PRICE operation came from the ppth partition of A and I . pp is set to zero when the basis is refactored. A PRICE operation is defined to be the process by which a nonbasic variable is selected to become a new superbasic. The selected variable is denoted by jq. Variable jq often becomes basic immediately. Otherwise it remains superbasic, unless it reaches its opposite bound and becomes nonbasic again. If Partial price is in effect, variable jq is selected from *A*pp or *I*pp , the ppth segments of the constraint matrix (A I).
rg	In Phase 1, 2 or 3, this is d_j, the reduced cost (reduced gradient) of the variable jq selected by PRICE at the start of the present iteration. Algebraically, $d_j = g_j - \pi^T a_j$ for j = jq, where g_j is the gradient of the current objective function, π is the vector of dual variables for the problem (or LC subproblem), and a_j is the jth column of the current (A I). In Phase 4, rg is the largest reduced gradient among the superbasic variables.
+sbs	The variable jq selected by PRICE to be added to the superbasic set.
-sbs	The variable chosen to leave the set of superbasics. It has become basic if the entry under -bs is nonzero; otherwise it has become nonbasic.
-bs	The variable removed from the basis (if any) to become nonbasic.
step	The step length a taken along the current search direction p. The variables x have just been changed to $x + ap$.
pivot	If column aq replaces the rth column of the basis B, pivot is the rth element of a vector y satisfying $By = aq$. Wherever possible, step is chosen to avoid extremely small values of pivot (because they cause the basis to be nearly singular). In rare cases, it may be necessary to increase the Pivot tolerance to exclude very small elements of y from consideration during the computation of step.
ninf	The number of infeasibilities *before* the present iteration. This number decreases monotonically.
sinf,objective	If ninf > 0, this is sinf, the sum of infeasibilities before the present iteration. It usually decreases at each nonzero step, but if ninf decreases by 2 or more, sinf may occasionally increase. Otherwise it is the value of the current objective function *after* the present iteration. For linear programs, it means the true linear objective function. For problems with linear constraints, it means the sum of the linear objective and the value returned by subroutine funobj. For problems with nonlinear constraints, it is the quantity just described if Lagrangian = No; otherwise it is the value of the augmented Lagrangian for the current major iterations (which tends to the true objective as convergence is approached).

L	The number of nonzeros representing the basis factor *L*. Immediately after a basis factorization $B = LU$, this is lenL, the number of subdiagonal elements in the columns of a lower triangular matrix. Further nonzeros are added to L when various columns of *B* are later replaced. (Thus, L increases monotonically.)
U	The number of nonzeros in the basis factor U . Immediately after a basis factorization, this is lenU, the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. As columns of *B* are replaced, the matrix *U* is maintained explicitly (in sparse form). The value of U may fluctuate up or down; in general it will tend to increase.
ncp	The number of compressions required to recover storage in the data structure for U . This includes the number of compressions needed during the previous basis factorization. Normally ncp should increase very slowly. If not, the amount of workspace available to MINOS should be increased by a significant amount. As a suggestion, the work array z(*) should be extended by 2(L + U) elements.

The following items are printed if the problem is nonlinear or if the superbasic set is non-empty (i.e., if the current solution is not a vertex).

Label	**Description**
nobj	The number of times subroutine funobj has been called.
ncon	The number of times subroutine funcon has been called.
nsb	The current number of superbasic variables.
Hmod	An indication of the type of modifications made to the triangular matrix *R* that is used to approximate the reduced Hessian matrix. Two integers i_1 and i_2 are shown. They will remain zero for linear problems. If $i_1 = 1$, a BFGS quasi-Newton update has been made to *R*, to account for a move within the current subspace. (This will not occur if the solution is infeasible.) If $i_2 = 1$, *R* has been modified to account for a change in basis. This will sometimes occur even if the solution is infeasible (if a feasible point was obtained at some earlier stage). Both updates are implemented by triangularizing the matrix $R + vw^T$ for some vectors *v* and *w*. If an update fails for numerical reasons, i_1 or i_2 will be set to 2, and the resulting *R* will be nearly singular. (However, this is highly unlikely.)
cond(H)	An estimate of the condition number of the reduced Hessian. It is the square of the ratio of the largest and smallest diagonals of the upper triangular matrix *R*-a lower bound on the condition number of the matrix *RTR* that approximates the reduced Hessian. cond(H) gives a rough indication of whether or not the optimization procedure is having difficulty. The reduced-gradient algorithm will make slow progress if cond(H) becomes as large as 10^8 , and will probably fail to find a better solution if cond(H) reaches 10^{12} or more. To guard against high values of cond(H), attention should be given to the scaling of the variables and the constraints. In some cases it may be necessary to add upper or lower bounds to certain variables to keep them a reasonable distance from singularities in the nonlinear functions or their derivatives.
conv	A set of four logical variables C_1 , C_2 , C_3 , C_4 that are used to determine when to discontinue optimization in the current subspace (Phase 4) and consider releasing a nonbasic variable from its bound (the PRICE operation of Phase 3). Let rg be the norm of the reduced gradient, as described above. The meaning of the variables C_j is as follows: C_1 is true if the change in *x* was sufficiently small; C_2 is true if the change in the objective was sufficiently small; C_3 is true if rg is smaller than some loose tolerance TOLRG; C_4 is true if rg is smaller than some tighter tolerance. The test used is of the form *if* (C_1 *and* C_2 *and* C_3) *or* C_4 *then go to Phase 3.* At present, tolrg = *t*|dj|, where *t* is the Subspace tolerance (nominally 0.5) and dj is the reduced- gradient norm at the most recent Phase 3 iteration. The "tighter tolerance" is the maximum of 0.1 tolrg and $10^{-7}\|\pi\|$. *Only the tolerance* t *can be altered at run-time.*

Crash statistics

The following items are output to the PRINT file when no warm start is used. They refer to the number of columns that the CRASH procedure selects during several passes through A while searching for a triangular basis matrix.

Label	Description
Slacks	is the number of slacks selected initially.
Free cols	is the number of free columns in the basis, including those whose bounds are rather far apart.
Preferred	is the number of "preferred" columns in the basis (i.e., hs(j) = 3 for some $j <= n$). It will be a subset of the columns for which hs(j) = 3 was specified.
Unit	is the number of unit columns in the basis.
Double	is the number of columns in the basis containing 2 nonzeros.
Triangle	is the number of triangular columns in the basis with 3 or more nonzeros.
Pad	is the number of slacks used to pad the basis (to make it a nonsingular triangle).

Basis factorization statistics

If Printlevel => 1, the following items are output to the PRINT file whenever the basis B or the rectangular matrix $B_S = (B S)^T$ is factorized. Note that B_S may be factorized at the start of just some of the major iterations. It is immediately followed by a factorization of B itself.

Gaussian elimination is used to compute a sparse LU factorization of B or BS , where PLP^T and PUQ are lower and upper triangular matrices for some permutation matrices P and Q.

Label	Description
Factorize	The number of factorizations since the start of the run.
Demand	A code giving the reason for the present factorization.
Itn	The current iteration number.
Nonlin	The number of nonlinear variables in the current basis B.
Linear	The number of linear variables in B.
Slacks	The number of slack variables in B.
m	The number of rows in the matrix being factorized (B or BS).
n	The number of columns in the matrix being factorized. Preceded by "=" if the matrix is B; by ">" if it is BS .
Elems	The number of nonzero elements in B or BS .
Amax	The largest nonzero in B or BS .
Density	The density of the matrix (percentage of nonzeros).
Merit	The average Markowitz merit count for the elements chosen to be the diagonals of PUQ. Each merit count is defined to be (c - 1)(r - 1) where c and r are the number of nonzeros in the column and row containing the element at the time it is selected to be the next diagonal. Merit is the average of m such quantities. It gives an indication of how much work was required to preserve sparsity during the factorization.
lenL	The number of nonzeros in the factor L.
L+U	The number of nonzeros in both L and U .
Cmprssns	The number of times the data structure holding the partially factored matrix needed to be compressed to recover unused storage. Ideally this number should be zero. If it is more than 3 or 4, the amount of workspace available to MINOS should be increased for efficiency.
Incres	The percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B or BS .

Utri	The size of the "backward triangle" in *B* or *BS* . These top rows of *U* come directly from the matrix.
lenU	The number of nonzeros in the factor *U* .
Ltol	The maximum allowed size of nonzeros in *L*. Usually equal to the LU factor tolerance.
Umax	The maximum nonzero in *U* .
Ugrwth	The ratio Umax / Amax.
Ltri	The size of the "forward triangle" in *B* or *BS* . These initial columns of *L* come directly from the matrix.
dense1	is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.
Lmax	The maximum nonzero in *L* (no larger than Ltol).
Akmax	The maximum nonzero arising during the factorization. (Printed only if Threshold Complete Pivoting is in effect.)
Agrwth	The ratio Akmax / Amax. (Printed only if Threshold Complete Pivoting is in effect.)
bump	The number of columns of *B* or *BS* excluding Utri and Ltri.
dense2	The number of columns remaining when the density of the basis matrix being factorized reached 0.6.
DUmax	The largest diagonal of *U* (really *P U Q*).
DUmin	The smallest diagonal of *U* .
condU	The ratio DUmax/DUmin. As long as Ltol is not large (say 10.0 or less), condU is an estimate of the condition number of *B*. If this number is extremely large, the basis is nearly singular and some numerical difficulties might occur. (However, an effort is made to avoid near-singularity by using slacks to replace columns of *B* that would have made Umin extremely small. Messages are issued to this effect, and the modified basis is refactored.)

EXIT conditions

When the solution procedure terminates, an EXIT -- message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

The number associated with each EXIT is the output value of the integer variable inform.

The following messages arise when the SPECS file is found to contain no further problems.

```
-2       EXIT -- input error.
```

MINOS encountered end-of-file or an endrun card before finding a SPECS file.

Otherwise, the SPECS file may be empty, or cards containing the keywords Skip or Endrun may imply that all problems should be ignored.

```
-1    ENDRUN
```

This message is printed at the end of a run if MINOS terminates of its own accord. Otherwise, the operating system will have intervened for one of many possible reasons (excess time, missing file, arithmetic error in user routines, etc.).

The following messages arise when a solution exists (though it may not be optimal).

```
0      EXIT -- optimal solution  found
```

This is the message we all hope to see! It is certainly preferable to every other message, and we naturally want to believe what it says, because this is surely one situation where *the computer knows best*. There may be cause for celebration if the objective function has reached an astonishing new high (or low).

In all cases, a distinct level of caution is in order, even if it can wait until next morning. For example, if the objective value *is* much better than expected, we may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is

one of the more difficult tasks for a model builder. It is good practice in the function subroutines to print any data that is input during the first entry.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Always specify a good starting point if possible, especially for nonlinear variables, and include reasonable upper and lower bounds on the variables to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as well as they can.

One other caution about "Optimal solution"s. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. Max Primal infeas refers to the largest bound infeasibility and which variable (or slack) is involved. If it is too large, consider restarting with a smaller Feasibility tolerance (say 10 times smaller) and perhaps Scale option 0.

Similarly, Max Dual infeas indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

MaxDualinfeas/Normofpi = 10^{-d},

then the objective function would probably change in the dth significant digit if optimization could be continued. If d seems too large, consider restarting with smaller Optimality tolerances.

Finally, Nonlinear constraint violn shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller Row tolerance.

```
1      EXIT -- the  problem  is infeasible
```

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax+s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the Feasibility tolerance are ignored, but at least one component of x or s violates a bound by more than the tolerance.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each linearly constrained (LC) subproblem, MINOS is prepared to relax the bounds on the slacks associated with nonlinear rows.

If an LC subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), MINOS enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities-suitably weighted using the Elastic weight parameter. In elastic mode, some of the bounds on the nonlinear rows "elastic"-i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, MINOS will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, MINOS would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though MINOS locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined

in such a way that feasible points are known to exist when the constraints are linearized.

```
2      EXIT -- the  problem  is unbounded      (or badly  scaled)
      EXIT -- violation limit exceeded -- the  problem  may  be unbounded
```

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. A message prior to the EXIT message will give the index of the nonbasic variable. Consider adding an upper or lower bound to the variable. Also, examine the constraints that have nonzeros in the associated column, to see if they have been formulated as intended.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the Scale option.

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the Unbounded parameters, the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the Violation limit.

```
3     EXIT -- major  iteration limit exceeded EXIT -- minor  iteration limit exceeded EXIT -- too  many  iterations
```

Either the Iterations limit or the Major iterations limit was exceeded before the required solution could be found. Check the iteration log to be sure that progress was being made. If so, restart the run using *WarmDefSOL* at the end of the run.

```
4      EXIT -- requested  accuracy  could  not  be achieved
```

A feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but MINOS is within 10-2 of satisfying the Major optimality tolerance. Check that the Major optimality tolerance is not too small.

```
5      EXIT -- the  superbasics limit is too  small:      nnn
```

The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already nnn superbasics (and no room for any more).

In general, raise the Superbasics limit *s* by a reasonable amount, bearing in mind the storage needed for the reduced Hessian (about 1 *s*2 double words).

```
6      EXIT --  constraint and objective values  could  not  be calculated
```

This exit occurs if a value mode = -1 is set during some call to funobj or funcon. MINOS assumes that you want the problem to be abandoned forthwith.

In some environments, this exit means that your subroutines were not successfully linked to MINOS. If the default versions of funobj and funcon are ever called, they issue a warning message and then set mode to terminate the run.

```
7      EXIT -- subroutine funobj seems to  be giving incorrect  gradients
```

A check has been made on some individual elements of the objective gradient array at the first point that satisfies the linear constraints. At least one component gObj(*j*) is being set to a value that disagrees markedly with a forward-difference estimate of *?f /?xj* . (The relative difference between the computed and estimated values is 1.0 or

more.) This exit is a safeguard, since MINOS will usually fail to make progress when the computed gradients are seriously inaccurate. In the process it may expend considerable effort before terminating with EXIT 9 below.

Check the function and gradient computation *very carefully* in funobj. A simple omission (such as forgetting to divide fObj by 2) could explain everything. If fObj or gObj(*j*) is very large, then give serious thought to scaling the function or the nonlinear variables.

If you feel *certain* that the computed gObj(*j*) is correct (and that the forward-difference estimate is therefore wrong), you can specify Verify level 0 to prevent individual elements from being checked. However, the optimization procedure may have difficulty.

```
8      EXIT -- subroutine funcon  seems to  be giving incorrect  gradients
```

This is analogous to the preceding exit. At least one of the computed Jacobian elements is significantly different from an estimate obtained by forward-differencing the constraint vector $F(x)$. Follow the advice given above, trying to ensure that the arrays fCon and gCon are being set correctly in funcon.

```
9      EXIT -- the  current point cannot  be improved  upon
```

Several circumstances could lead to this exit.

1. Subroutines funobj or funcon could be returning accurate function values but inaccurate gradients (or vice versa). This is the most likely cause. Study the comments given for EXIT 7 and 8, and do your best to ensure that the coding is correct.
2. The function and gradient values could be consistent, but their precision could be too low. For example, accidental use of a real data type when double precision was intended would lead to a relative function precision of about 10^{-6} instead of something like 10^{-15}. The default Optimality tolerance of 10^{-6} would need to be raised to about 10^{-3} for optimality to be declared (at a rather suboptimal point). Of course, it is better to revise the function coding to obtain as much precision as economically possible.
3. If function values are obtained from an expensive iterative process, they may be accurate to rather few significant figures, and gradients will probably not be available. One should specify

Function precision	t
Major optimality tolerance	$\sqrt{t}$

but even then, if t is as large as 10^{-5} or 10^{-6} (only 5 or 6 significant figures), the same exit condition may occur. At present the only remedy is to increase the accuracy of the function calculation.

```
10   EXIT -- cannot  satisfy the  general  constraints
```

An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

This probably means that the current basis is very ill-conditioned. Try Scale option 1 if scaling has not yet been used and there are some linear constraints and variables.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Consult the description of Umax, Umin and Growth in #Basis factorization statistics, and set the LU factor tolerance to 2.0 (or possibly even smaller, but not less than 1.0).

```
12   EXIT -- terminated from  subroutine s1User
```

The user has set the value iAbort = 1 in subroutine s1User. MINOS assumes that you want the problem to be abandoned forthwith.

If the following exits occur during the *first* basis factorization, the primal and dual variables x and pi will have their original input values. BASIS files will be saved if requested, but certain values in the printed solution will not be meaningful.

```
20   EXIT -- not  enough integer/real storage  for the  basis  factors
```

The main integer or real storage array iw(*) or rw(*) is apparently not large enough for this problem. The routine declaring iw and rw should be recompiled with a larger dimensions for those arrays. The new values should also be assigned to leniw and lenrw.

An estimate of the additional storage required is given in messages preceding the exit.

```
21   EXIT -- error in basis  package
```

A preceding message will describe the error in more detail. One such message says that the current basis has more than one element in row *i* and column *j*. This could be caused by a corresponding error in the input parameters a(*), ha(*), and ka(*).

```
22   EXIT -- singular basis  after nnn factorization attempts
```

This exit is highly unlikely to occur. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix *U* were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized, but singularity persists. This must mean that the problem is badly scaled, or the LU factor tolerance is too much larger than 1.0.

If the following messages arise, either an OLD BASIS file could not be loaded properly, or some fatal system error has occurred. New BASIS files cannot be saved, and there is no solution to print. The problem is abandoned.

```
30   EXIT -- the  basis  file dimensions  do not  match this problem
```

On the first line of the OLD BASIS file, the dimensions labeled m and n are different from those associated with the problem that has just been defined. You have probably loaded a file that belongs to another problem.

Remember, if you have added rows or columns to a(*), ha(*) and ka(*), you will have to alter m and n *and* the map beginning on the third line (a hazardous operation). It may be easier to restart with a PUNCH or DUMP file from an earlier version of the problem.

```
31   EXIT -- the  basis  file state vector does not  match this problem
```

For some reason, the OLD BASIS file is incompatible with the present problem, or is not consistent within itself. The number of basic entries in the state vector (i.e., the number of 3's in the map) is not the same as m on the first line, or some of the 2's in the map did not have a corresponding "$j\ x_j$" entry following the map.

```
32   EXIT -- system error.  Wrong no.  of  basic  variables:   nnn
```

This exit should never happen. It may indicate that the wrong MINOS source files have been compiled, or incorrect parameters have been used in the call to subroutine minoss.

Check that all integer variables and arrays are declared integer in your calling program (including those beginning with h!), and that all "real" variables and arrays are declared consistently. (They should be double precision on most machines.)

The following messages arise if additional storage is needed to allow optimization to begin. The problem is abandoned.

```
42   EXIT -- not  enough 8-character storage  to  start  solving the  problem
```

The main character storage array cw(*) is not large enough.

```
43   EXIT -- not  enough integer storage  to  start  solving the  problem
```

The main integer storage array iw(*) is not large enough to provide workspace for the optimization procedure. See the advice given for Exit 20.

```
44   EXIT -- not  enough real storage  to  start  solving the  problem
```

The main storage array rw(*) is not large enough to provide workspace for the optimization procedure. Be sure that the Superbasics limit is not unreasonably large. Otherwise, see the advice for EXIT 20.

Solution output

At the end of a run, the final solution is output to the PRINT file in accordance with the Solution keyword. Some header information appears first to identify the problem and the final state of the optimization procedure. A ROWS section and a COLUMNS section then follow, giving one line of information for each row and column. The format used is similar to certain commercial systems, though there is no industry standard.

An example of the printed solution is given in #top. In general, numerical values are output with format f16.5. The maximum record length is 111 characters, including the first (carriage-control) character.

To reduce clutter, a dot "." is printed for any numerical value that is exactly zero. The values ±1 are also printed specially as 1.0 and -1.0. Infinite bounds ($\pm 10^{20}$ or larger) are printed as None.

Note : If two problems are the same except that one minimizes an objective $f(x)$ and the other maximizes $-f(x)$, their solutions will be the same but the signs of the dual variables π_i and the reduced gradients d_j will be reversed.

The ROWS section

General linear constraints take the form $l \leq Ax \leq u$. The *i*th constraint is therefore of the form

$$\alpha \leq a^T x \leq \beta,$$

and the value of $a^T x$ is called the *row activity*. Internally, the linear constraints take the form $Ax - s = 0$, where the slack variables s should satisfy the bounds $l \leq s \leq u$. For the *i*th "row", it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. Slacks may be basic or nonbasic (but not superbasic).

Nonlinear constraints $alpha \leq F_i(x) + a^T x \leq \beta$ are treated similarly, except that the row activity and degree of infeasibility are computed directly from $Fi(x) + a^T x$ rather than from s_i.

Label	Description
Number	The value $n + i$. This is the internal number used to refer to the *i*th slack in the iteration log.
Row	The name of the *i*th row.
State	The state of the *i*th row relative to the bounds α and β. The various states possible are as follows. LL — The row is at its lower limit, α. UL — The row is at its upper limit, β. EQ — The limits are the same ($\alpha = \beta$). BS — The constraint is not binding. s_i is basic. A key is sometimes printed before the State to give some additional information about the state of the slack variable.

	<table><tr><td>A</td><td>Alternative optimum possible. The slack is nonbasic, but its reduced gradient is essen- tially zero. This means that if the slack were allowed to start moving from its current value, there would be no change in the objective function. The values of the basic and superbasic variables might change, giving a genuine alternative solution. The values of the dual variables might also change.</td></tr><tr><td>D</td><td>Degenerate. The slack is basic, but it is equal to (or very close to) one of its bounds.</td></tr><tr><td>I</td><td>Infeasible. The slack is basic and is currently violating one of its bounds by more than the Feasibility tolerance.</td></tr><tr><td>N</td><td>Not precisely optimal. The slack is nonbasic. Its reduced gradient is larger than the Optimality tolerance. Note: If Scaleoption > 0, the tests for assigning A, D, I, N are made on the scaled problem, since the keys are then more likely to be meaningful.</td></tr></table>
Activity	The row value $a^T x$ (or $F_i(x) + a^T x$ for nonlinear rows).
Slack activity	The amount by which the row differs from its nearest bound. (For free rows, it is taken to be minus the Activity.)
Lower limit	α, the lower bound on the row.
Upper limit	β, the upper bound on the row.
Dual activity	The value of the dual variable *pi* , often called the shadow price (or simplex multiplier) for the *i*th constraint. The full vector *p* always satisfies $B^T p = g_B$, where *B* is the current basis matrix and *gB* contains the associated gradients for the current objective function.
I	The constraint number, *i*.

The COLUMNS section

Here we talk about the "column variables" x_j , $j = 1 : n$. We assume that a typical variable has bounds $\alpha \leq x_j \leq \beta$.

Label	Description
Number	The column number, *j*. This is the internal number used to refer to x_j in the iteration log.
Column	The name of x_j .
State	The state of *xj* relative to the bounds *a* and *ß*. The various states possible are as follows. <table><tr><td>LL</td><td>x_j is nonbasic at its lower limit, α.</td></tr><tr><td>UL</td><td>x_j is nonbasic at its upper limit, β.</td></tr><tr><td>EQ</td><td>x_j is nonbasic and fixed at the value $\alpha = \beta$.</td></tr><tr><td>FR</td><td>x_j is nonbasic at some value strictly between its bounds: $\alpha < x_j < \beta$.</td></tr><tr><td>BS</td><td>x_j is basic. Usually $\alpha < x_j < \beta$.</td></tr><tr><td>SBS</td><td>x_j is superbasic. Usually $\alpha < x_j < \beta$.</td></tr></table> A key is sometimes printed before the State to give some additional information about the state of x_j.

	<table><tr><td>A</td><td>Alternative optimum possible. The variable is nonbasic, but its reduced gradient is essentially zero. This means that if x_j were allowed to start moving from its current value, there would be no change in the objective function. The values of the basic and superbasic variables might change, giving a genuine alternative solution. The values of the dual variables might also change.</td></tr><tr><td>D</td><td>Degenerate. x_j is basic, but it is equal to (or very close to) one of its bounds.</td></tr><tr><td>I</td><td>Infeasible. x_j is basic and is currently violating one of its bounds by more than the Feasibility tolerance.</td></tr><tr><td>N</td><td>Not precisely optimal. xj is nonbasic. Its reduced gradient is larger than the Optimality tolerance . Note: If Scaleoption > 0, the tests for assigning A, D, I, N are made on the scaled problem, since the keys are then more likely to be meaningful.</td></tr></table>
Activity	The value of the variable x_j.
Obj Gradient	g_j, the jth component of the gradient of the (linear or nonlinear) objective function. (If any x_j is infeasible, g_j is the gradient of the sum of infeasibilities.)
Lower limit	α, the lower bound on x_j.
Upper limit	β, the upper bound on x_j.
Reduced gradient	The reduced gradient $d_j = g_j - \pi^T a_j$, where a_j is the jth column of the constraint matrix (or the jth column of the Jacobian at the start of the final major iteration).
M+J	The value $m + j$.

The SUMMARY file

If Summaryfile > 0, the following information is output to the SUMMARY file. (It is a brief form of the PRINT file.) All output lines are less than 72 characters.

- The Begin line from the SPECS file, if any.
- The basis file loaded, if any.
- A brief Major iteration log.
- A brief Minor iteration log.
- The EXIT condition and a summary of the final solution.

The following SUMMARY file is from example problem t6wood using Print level 0 and Major damping parameter 0.5.

```
     ==============================
     M I N O S  5.51     (Nov 2002)
     ==============================

Begin t6wood   (WOPLANT test problem; optimal obj = -15.55716)

Name   WOPLANT
===>  Note:  row  OBJ       selected as linear part of objective.
Rows          9
Columns      12
Elements     73

Scale option  2,    Partial price   1
```

```
Itn      0 -- linear constraints satisfied.

This is problem t6wood.  Derivative level =  3

funcon  sets      36   out of      50   constraint gradients.

Major minor  step     objective  Feasible Optimal  nsb   ncon penalty BSwap
    1     0T 0.0E+00  0.00000E+00 5.9E-01 1.1E+01    0      4 1.0E+00     0
    2    22  5.0E-01 -1.56839E+01 2.7E-01 1.6E+01    3     47 1.0E+00     0
    3    10  6.0E-01 -1.51527E+01 1.5E-01 9.9E+00    2     68 1.0E+00     2
    4    21  5.7E-01 -1.53638E+01 6.4E-02 3.6E+00    3    113 1.0E+00     1
    5    15  1.0E+00 -1.55604E+01 2.7E-02 1.4E-01    3    144 1.0E+00     0
    6     5  1.0E+00 -1.55531E+01 6.4E-03 2.2E-01    3    154 1.0E+00     0
    7     4  1.0E+00 -1.55569E+01 3.1E-04 7.0E-04    3    160 1.0E-01     0
    8     2  1.0E+00 -1.55572E+01 1.6E-08 1.1E-04    3    163 1.0E-02     0
    9     1  1.0E+00 -1.55572E+01 5.1E-14 2.2E-06    3    165 1.0E-03     0

EXIT -- optimal solution found

Problem name                 WOPLANT
No. of iterations                 80   Objective value     -1.5557160112E+01
No. of major iterations            9   Linear objective    -1.5557160112E+01
Penalty parameter           0.000100   Nonlinear objective  0.0000000000E+00
No. of calls to funobj             0   No. of calls to funcon            165
No. of superbasics                 3   No. of basic nonlinears             6
No. of degenerate steps            0   Percentage                       0.00
Norm of x   (scaled)         9.8E-01   Norm of pi  (scaled)          1.8E+02
Norm of x                    3.2E+01   Norm of pi                    1.6E+01
Max Prim inf(scaled)       0 0.0E+00   Max Dual inf(scaled)        1 2.2E-06
Max Primal infeas          0 0.0E+00   Max Dual infeas             1 5.8E-08
Nonlinear constraint violn   5.1E-14



Solution printed on file   9

funcon called with nstate =   2

Time for MPS input                          0.00 seconds
Time for solving problem                    0.04 seconds
Time for solution output                    0.00 seconds
Time for constraint functions               0.00 seconds
Time for objective function                 0.00 seconds
```

The following information is output to the PRINT file during the solution process. The longest line of output is 124 characters.

- A listing of the SPECS file, if any.

- The selected options.
- An estimate of the storage needed and the amount available.
- Some statistics about the problem data.
- The storage available for the LU factors of the basis matrix.
- A log from the scaling procedure, if Scaleoption > 0.
- Notes about the initial basis obtained from CRASH or a BASIS file.
- The major iteration log.
- The minor iteration log.
- Basis factorization statistics.
- The EXIT condition and some statistics about the solution obtained.
- The printed solution, if requested.

The last five items are described in the following sections.

Article Sources and Contributors

MINOS *Source*: http://tomwiki.com/index.php?oldid=2373 *Contributors*: Elias

MINOS Solver Options *Source*: http://tomwiki.com/index.php?oldid=2376 *Contributors*: Elias

MINOS File Output *Source*: http://tomwiki.com/index.php?oldid=1066 *Contributors*: Elias