NAG Toolbox for MATLAB

e04nk

1 Purpose

e04nk solves sparse linear programming or quadratic programming problems.

2 Syntax

```
[ns, xs, istate, miniz, minz, ninf, sinf, obj, clamda, user, lwsav,
iwsav, rwsav, ifail] = e04nk(n, m, iobj, ncolh, qphx, a, ha, ka, bl, bu,
start, names, crname, ns, xs, istate, leniz, lenz, lwsav, iwsav, rwsav,
'nnz', nnz, 'nname', nname, 'user', user)
```

Before calling e04nk, or the option setting function e04nm, e04wb **must** be called.

3 Description

e04nk is designed to solve a class of quadratic programming problems that are assumed to be stated in the following general form:

$$\underset{x \in R^n}{\text{minimize}} f(x) \qquad \text{subject to} \qquad l \leq \left\{ \begin{array}{c} x \\ Ax \end{array} \right\} \leq u, \tag{1}$$

where x is a set of variables, A is an m by n matrix and the objective function $f(x)$ may be specified in a variety of ways depending upon the particular problem to be solved. The optional parameter **Maximize** may be used to specify an alternative problem in which $f(x)$ is maximized. The possible forms for $f(x)$ are listed in Table 1, in which the prefixes FP, LP and QP stand for 'feasible point', 'linear programming' and 'quadratic programming' respectively, c is an n element vector and H is the n by n second-derivative matrix $\nabla^2 f(x)$ (the *Hessian matrix*).

Problem type	Objective function $f(x)$	Hessian matrix H
FP	Not applicable	Not applicable
LP	$c^{\mathrm{T}}x$	Not applicable
QP	$c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$	Symmetric positive semi-definite

Table 1

For LP and QP problems, the unique global minimum value of $f(x)$ is found. For FP problems, $f(x)$ is omitted and the function attempts to find a feasible point for the set of constraints. For QP problems, you must also provide a (sub)program that computes Hx for any given vector x. (H need not be stored explicitly.) If H is the zero matrix, the function will still solve the resulting LP problem; however, this can be accomplished more efficiently by setting **ncolh** $= 0$ (see Section 5).

The defining feature of a *convex* QP problem is that the matrix H must be *positive semi-definite*, i.e., it must satisfy $x^{\mathrm{T}}Hx \geq 0$ for all x. Otherwise, $f(x)$ is said to be *nonconvex* and it may be more appropriate to call e04ug instead.

e04nk is intended to solve large-scale linear and quadratic programming problems in which the constraint matrix A is *sparse* (i.e., when the number of zero elements is sufficiently large that it is worthwhile using algorithms which avoid computations and storage involving zero elements). The function also takes advantage of sparsity in c. (Sparsity in H can be exploited in the (sub)program that computes Hx.) For problems in which A can be treated as a *dense* matrix, it is usually more efficient to use e04mf, e04nc or e04nf.

The upper and lower bounds on the m elements of Ax are said to define the *general constraints* of the problem. Internally, e04nk converts the general constraints to equalities by introducing a set of *slack variables* s, where $s = (s_1, s_2, \ldots, s_m)^{\mathrm{T}}$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$, together with the bounded slack $5 \leq s_1 \leq +\infty$. The problem defined by

(1) can therefore be re-written in the following equivalent form:

$$\underset{x \in R^n, s \in R^m}{\text{minimize}} f(x) \qquad \text{subject to} \qquad Ax - s = 0, \qquad l \leq \begin{Bmatrix} x \\ s \end{Bmatrix} \leq u.$$

Since the slack variables s are subject to the same upper and lower bounds as the elements of Ax, the bounds on Ax and x can simply be thought of as bounds on the combined vector (x, s). (In order to indicate their special role in QP problems, the original variables x are sometimes known as 'column variables', and the slack variables s are known as 'row variables'.)

Each LP or QP problem is solved using an *active-set* method. This is an iterative procedure with two phases: a *feasibility phase*, in which the sum of infeasibilities is minimized to find a feasible point; and an *optimality phase*, in which $f(x)$ is minimized by constructing a sequence of iterations that lies within the feasible region.

A constraint is said to be *active* or *binding* at x if the associated element of either x or Ax is equal to one of its upper or lower bounds. Since an active constraint in Ax has its associated slack variable at a bound, the status of both simple and general upper and lower bounds can be conveniently described in terms of the status of the variables (x, s). A variable is said to be *nonbasic* if it is temporarily fixed at its upper or lower bound. It follows that regarding a general constraint as being *active* is equivalent to thinking of its associated slack as being *nonbasic*.

At each iteration of an active-set method, the constraints $Ax - s = 0$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = 0,$$

where x_N consists of the nonbasic elements of (x, s) and the *basis matrix* B is square and nonsingular. The elements of x_B and x_S are called the *basic* and *superbasic* variables respectively; with x_N they are a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie somewhere between their upper or lower bounds, while the nonbasic variables will be equal to one of their bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the objective function (or sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = 0$. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero for FP and LP problems.

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

Associated with each of the m equality constraints $Ax - s = 0$ is a *dual variable* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j (also known as a *reduced cost*). The reduced gradients for the variables x are the quantities $g - A^{\mathrm{T}}\pi$, where g is the gradient of the QP objective function; and the reduced gradients for the slack variables s are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds and $d_j = 0$ for all superbasic variables. In practice, an *approximate* QP solution is found by slightly relaxing these conditions on d_j (see the description of the optional parameter **Optimality Tolerance**).

The process of computing and comparing reduced gradients is known as *pricing* (a term first introduced in the context of the simplex method for linear programming). To 'price' a nonbasic variable x_j means that the reduced gradient d_j associated with the relevant active upper or lower bound on x_j is computed via the formula $d_j = g_j - a^{\mathrm{T}}\pi$, where a_j is the jth column of $\begin{pmatrix} A & -I \end{pmatrix}$. (The variable selected by such a process and the corresponding value of d_j (i.e., its reduced gradient) are the quantities `+S` and `dj` in the monitoring file output; see Section 12.) If A has significantly more columns than rows (i.e., $n \gg m$), pricing can be computationally expensive. In this case, a strategy known as *partial pricing* can be used to compute and compare only a subset of the d_j's.

e04nk is based on SQOPT, which is part of the SNOPT package described in Gill *et al.* 2002, which in turn utilizes functions from the MINOS package (see Murtagh and Saunders 1995). It uses stable numerical methods throughout and includes a reliable basis package (for maintaining sparse LU factors of the basis matrix B), a practical anti-degeneracy procedure, efficient handling of linear constraints and bounds on the variables (by an active-set strategy), as well as automatic scaling of the constraints. Further details can be found in Section 10.

4 References

Fourer R 1982 Solving staircase linear programs by the simplex method *Math. Programming* **23** 274–313

Gill P E and Murray W 1978 Numerically stable methods for quadratic programming *Math. Programming* **14** 349–372

Gill P E, Murray W and Saunders M A 2002 *SNOPT: An SQP Algorithm for Large-scale Constrained Optimization* **12** 979–1006 SIAM J. Optim.

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

Gill P E, Murray W, Saunders M A and Wright M H 1989 A practical anti-cycling procedure for linearly constrained optimization *Math. Programming* **45** 437–474

Gill P E, Murray W, Saunders M A and Wright M H 1991 Inertia-controlling methods for general quadratic programming *SIAM Rev.* **33** 1–36

Hall J A J and McKinnon K I M 1996 The Simplest Examples where the Simplex Method Cycles and Conditions where EXPAND Fails to Prevent Cycling *Report MS* 96–100 Department of Mathematics and Statistics, University of Edinburgh

Murtagh B A and Saunders M A 1995 MINOS 5.4 Users' Guide *Report SOL 83-20R* Department of Operations Research, Stanford University

5 Parameters

5.1 Compulsory Input Parameters

1: **n – int32 scalar**

n, the number of variables (excluding slacks). This is the number of columns in the linear constraint matrix A.

Constraint: $\mathbf{n} \geq 1$.

2: **m – int32 scalar**

m, the number of general linear constraints (or slacks). This is the number of rows in A, including the free row (if any; see **iobj**).

Constraint: $\mathbf{m} \geq 1$.

3: **iobj – int32 scalar**

If $\mathbf{iobj} > 0$, row **iobj** of A is a free row containing the nonzero elements of the vector c appearing in the linear objective term $c^{\mathrm{T}}x$.

If $\mathbf{iobj} = 0$, there is no free row, i.e., the problem is either an FP problem (in which case **iobj** must be set to zero), or a QP problem with $c = 0$.

Constraint: $0 \leq \mathbf{iobj} \leq \mathbf{m}$.

4: **ncolh – int32 scalar**

n_H, the number of leading nonzero columns of the Hessian matrix H. For FP and LP problems, **ncolh** must be set to zero.

Constraint: $0 \leq \mathbf{ncolh} \leq \mathbf{n}$.

5: **qphx – string containing name of m-file**

For QP problems, you must supply a version of **qphx** to compute the matrix product Hx. If H has zero rows and columns, it is most efficient to order the variables $x = \begin{pmatrix} y & z \end{pmatrix}^{\mathrm{T}}$ so that

$$Hx = \begin{pmatrix} H_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} H_1 y \\ 0 \end{pmatrix},$$

where the nonlinear variables y appear first as shown. For FP and LP problems, **qphx** will never be called by e04nk and hence **qphx** may be the string `'e54nku'`.

```
[hx, user] = qphx(nstate, ncolh, x, user)
```

Input Parameters

1: **nstate – int32 scalar**

If **nstate** $= 1$, e04nk is calling **qphx** for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

If **nstate** ≥ 2, e04nk is calling **qphx** for the last time. This parameter setting allows you to perform some additional computation on the final solution. In general, the last call to **qphx** is made with **nstate** $= 2 +$ **ifail** (see Section 6).

Otherwise, **nstate** $= 0$.

2: **ncolh – int32 scalar**

This is the same parameter **ncolh** as supplied to e04nk.

3: **x(ncolh) – double array**

The first **ncolh** elements of the vector x.

4: **user – Any MATLAB object**

qphx is called from e04nk with **user** as supplied to e04nk

Output Parameters

1: **hx(ncolh) – double array**

The product Hx.

2: **user – Any MATLAB object**

qphx is called from e04nk with **user** as supplied to e04nk

6: **a(nnz) – double array**

The nonzero elements of A, ordered by increasing column index. Note that elements with the same row and column indices are not allowed.

7: **ha(nnz) – int32 array**

ha(i) must contain the row index of the nonzero element stored in **a**(i), for $i = 1, 2, \ldots,$ **nnz**. Note that the row indices for a column may be supplied in any order.

Constraint: $1 \leq$ **ha**$(i) \leq$ **m**, for $i = 1, 2, \ldots,$ **nnz**.

8: **ka**$(\mathbf{n}+1)$ – **int32 array**

ka(j) must contain the index in **a** of the start of the jth column, for $j = 1, 2, \ldots, \mathbf{n}$. To specify the jth column as empty, set **ka**(j) = **ka**$(j+1)$. Note that the first and last elements of **ka** must be such that **ka**$(1) = 1$ and **ka**$(\mathbf{n}+1) = \mathbf{nnz} + 1$.

Constraints:

ka$(1) = 1$;
ka$(j) \geq 1$, for $j = 2, 3, \ldots, \mathbf{n}$;
ka$(\mathbf{n}+1) = \mathbf{nnz} + 1$;
$0 \leq \mathbf{ka}(j+1) - \mathbf{ka}(j) \leq \mathbf{m}$, for $j = 1, 2, \ldots, \mathbf{n}$.

9: **bl**$(\mathbf{n}+\mathbf{m})$ – **double array**

l, the lower bounds for all the variables and general constraints, in the following order. The first **n** elements of **bl** must contain the bounds on the variables x, and the next **m** elements the bounds for the general linear constraints Ax (or slacks s) and the free row (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set **bl**$(j) \leq -bigbnd$, where $bigbnd$ is the value of the optional parameter **Infinite Bound Size**. To specify the jth constraint as an *equality*, set **bl**(j) = **bu**$(j) = \beta$, say, where $|\beta| < bigbnd$. Note that the lower bound corresponding to the free row must be set to $-\infty$ and stored in **bl**$(\mathbf{n}+\mathbf{iobj})$.

Constraint: if $\mathbf{iobj} > 0$, **bl**$(\mathbf{n}+\mathbf{iobj}) \leq -bigbnd$.

(See also the description for **bu**.)

10: **bu**$(\mathbf{n}+\mathbf{m})$ – **double array**

u, the upper bounds for all the variables and general constraints, in the following order. The first **n** elements of **bl** must contain the bounds on the variables x, and the next **m** elements the bounds for the general linear constraints Ax (or slacks s) and the free row (if any). To specify a nonexistent upper bound (i.e., $u_j = +\infty$), set **bu**$(j) \geq bigbnd$. Note that the upper bound corresponding to the free row must be set to $+\infty$ and stored in **bu**$(\mathbf{n}+\mathbf{iobj})$.

Constraints:

if $\mathbf{iobj} > 0$, **bu**$(\mathbf{n}+\mathbf{iobj}) \geq bigbnd$;
bl$(j) \leq$ **bu**(j), for $j = 1, 2, \ldots, \mathbf{n}+\mathbf{m}$;
if **bl**(j) = **bu**$(j) = \beta$, $|\beta| < bigbnd$.

11: **start** – **string**

Indicates how a starting basis is to be obtained.

start = 'C'

An internal Crash procedure will be used to choose an initial basis matrix B.

start = 'W'

A basis is already defined in **istate** (probably from a previous call).

Constraint: **start** = 'C' or 'W'.

12: **names**(5) – **string array**

A set of names associated with the so-called MPSX form of the problem, as follows:

names(1) must contain the name for the problem (or be blank);

names(2) must contain the name for the free row (or be blank);

names(3) must contain the name for the constraint right-hand side (or be blank);

names(4) must contain the name for the ranges (or be blank);

names(5) must contain the name for the bounds (or be blank).

(These names are used in the monitoring file output; see Section 12.)

13: **crname(nname) – string array**

The optional column and row names, respectively.

If **nname** $= 1$, **crname** is not referenced and the printed output will use default names for the columns and rows.

If **nname** $=$ **n** $+$ **m**, the first **n** elements must contain the names for the columns and the next **m** elements must contain the names for the rows. Note that the name for the free row (if any) must be stored in **crname**(**n** $+$ **iobj**).

14: **ns – int32 scalar**

n_S, the number of superbasics. For QP problems, **ns** need not be specified if **start** = 'C', but must retain its value from a previous call when **start** = 'W'. For FP and LP problems, **ns** need not be initialized.

15: **xs(n + m) – double array**

The initial values of the variables and slacks (x, s). (See the description for **istate**.)

16: **istate(n + m) – int32 array**

If **start** = 'C', the first **n** elements of **istate** and **xs** must specify the initial states and values, respectively, of the variables x. (The slacks s need not be initialized.) An internal Crash procedure is then used to select an initial basis matrix B. The initial basis matrix will be triangular (neglecting certain small elements in each column). It is chosen from various rows and columns of $\begin{pmatrix} A & -I \end{pmatrix}$. Possible values for **istate**(j) are as follows:

istate(j)	**State of xs**(j) **during Crash procedure**
0 or 1	Eligible for the basis
2	Ignored
3	Eligible for the basis (given preference over 0 or 1)
4 or 5	Ignored

If nothing special is known about the problem, or there is no wish to provide special information, you may set **istate**(j) $= 0$ and **xs**(j) $= 0.0$, for $j = 1, 2, \ldots,$ **n**. All variables will then be eligible for the initial basis. Less trivially, to say that the jth variable will probably be equal to one of its bounds, set **istate**(j) $= 4$ and **xs**(j) $=$ **bl**(j) or **istate**(j) $= 5$ and **xs**(j) $=$ **bu**(j) as appropriate.

Following the Crash procedure, variables for which **istate**(j) $= 2$ are made superbasic. Other variables not selected for the basis are then made nonbasic at the value **xs**(j) if **bl**(j) $\leq$ **xs**(j) $\leq$ **bu**(j), or at the value **bl**(j) or **bu**(j) closest to **xs**(j).

If **start** = 'W', **istate** and **xs** must specify the initial states and values, respectively, of the variables and slacks (x, s). If e04nk has been called previously with the same values of **n** and **m**, **istate** already contains satisfactory information.

Constraints:

if **start** = 'C', $0 \leq$ **istate**(j) ≤ 5, for $j = 1, 2, \ldots,$ **n**;
if **start** = 'W', $0 \leq$ **istate**(j) ≤ 3, for $j = 1, 2, \ldots,$ **n** $+$ **m**.

17: **leniz – int32 scalar**

Constraint: **leniz** ≥ 1.

18: **lenz – int32 scalar**

Constraint: **lenz** ≥ 1.

The amounts of workspace provided (i.e., **leniz** and **lenz**) and required (i.e., **miniz** and **minz**) are (by default for e04nk) output on the current advisory message unit NADV (as defined by x04ab).

Since the minimum values of **leniz** and **lenz** required to start solving the problem are returned in **miniz** and **minz**, respectively, you may prefer to obtain appropriate values from the output of a preliminary run with **leniz** and **lenz** set to 1. (e04nk will then terminate with **ifail** $= 12$.)

19: **lwsav(20) – logical array**

20: **iwsav(380) – int32 array**

21: **rwsav(285) – double array**

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions e04wb, e04nk, e04nm.

5.2 Optional Input Parameters

1: **nnz – int32 scalar**

Default: The dimension of the arrays **a**, **ha**. (An error is raised if these dimensions are not equal.)

the number of nonzero elements in A.

Constraint: $1 \leq$ **nnz** $\leq$ **n** $\times$ **m**.

2: **nname – int32 scalar**

Default: The dimension of the array **crname**.

the number of column (i.e., variable) and row names supplied in **crname**.

nname $= 1$

There are no names. Default names will be used in the printed output.

nname $=$ **n** $+$ **m**

All names must be supplied.

Constraint: **nname** $= 1$ or **n** $+$ **m**.

3: **user – Any MATLAB object**

user is not used by e04nk, but is passed to **qphx**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

iz, z

5.4 Output Parameters

1: **ns – int32 scalar**

The final number of superbasics. This will be zero for FP and LP problems.

2: **xs(n + m) – double array**

The final values of the variables and slacks (x, s).

3: **istate(n + m) – int32 array**

The final states of the variables and slacks (x, s). The significance of each possible value of **istate**(j) is as follows:

istate(j)	**State of variable** j	**Normal value of xs**(j)
0	Nonbasic	**bl**(j)
1	Nonbasic	**bu**(j)
2	Superbasic	Between **bl**(j) and **bu**(j)
3	Basic	Between **bl**(j) and **bu**(j)

If **ninf** $= 0$, basic and superbasic variables may be outside their bounds by as much as the value of the optional parameter **Feasibility Tolerance**. Note that unless the **Scale Option** $= 0$ is specified, the optional parameter **Feasibility Tolerance** applies to the variables of the scaled problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled.

Very occasionally some nonbasic variables may be outside their bounds by as much as the optional parameter **Feasibility Tolerance**, and there may be some nonbasic variables for which **xs**(j) lies strictly between its bounds.

If **ninf** > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by **sinf** if **Scale Option** $= 0$).

4: **miniz – int32 scalar**

The minimum value of **leniz** required to start solving the problem. If **ifail** $= 12$, e04nk may be called again with **leniz** suitably larger than **miniz**. (The bigger the better, since it is not certain how much workspace the basis factors need.)

5: **minz – int32 scalar**

The minimum value of **lenz** required to start solving the problem. If **ifail** $= 13$, e04nk may be called again with **lenz** suitably larger than **minz**. (The bigger the better, since it is not certain how much workspace the basis factors need.)

6: **ninf – int32 scalar**

The number of infeasibilities. This will be zero if **ifail** $=$ 0 or 1.

7: **sinf – double scalar**

The sum of infeasibilities. This will be zero if **ninf** $= 0$. (Note that e04nk does *not* attempt to compute the minimum value of **sinf** if **ifail** $= 3$.)

8: **obj – double scalar**

The value of the objective function.

If **ninf** $= 0$, **obj** includes the quadratic objective term $\frac{1}{2}x^{\mathrm{T}}Hx$ (if any).

If **ninf** > 0, **obj** is just the linear objective term $c^{\mathrm{T}}x$ (if any).

For FP problems, **obj** is set to zero.

9: **clamda(n + m) – double array**

A set of Lagrange multipliers for the bounds on the variables and the general constraints. More precisely, the first **n** elements contain the multipliers (*reduced costs*) for the bounds on the variables, and the next **m** elements contain the multipliers (*shadow prices*) for the general linear constraints.

10: **user – Any MATLAB object**

user is not used by e04nk, but is passed to **qphx**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

11: **lwsav(20) – logical array**

12: **iwsav(380) – int32 array**

13: **rwsav(285) – double array**

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions e04wb, e04nk, e04nm.

14: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Note: e04nk may return useful information for one or more of the following detected errors or warnings.

ifail $= 1$

Weak solution found. The final x is not unique, although x gives the global minimum value of the objective function.

ifail $= 2$

The problem is unbounded (or badly scaled). The objective function is not bounded below in the feasible region.

ifail $= 3$

The problem is infeasible. The general constraints cannot all be satisfied simultaneously to within the value of the optional parameter **Feasibility Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$, where ϵ is the ***machine precision***).

ifail $= 4$

Too many iterations. The value of the optional parameter **Iteration Limit** (default value $= \max(50, 5(n + m))$) is too small.

ifail $= 5$

The reduced Hessian matrix $Z^{\mathrm{T}}HZ$ (see Section 10.2) exceeds its assigned dimension. The value of the optional parameter **Superbasics Limit** (default value $= \min(n_H + 1, n)$) is too small.

ifail $= 6$

The Hessian matrix H appears to be indefinite. This sometimes occurs because the values of the optional parameters **LU Factor Tolerance** (default value $= 100.0$) and **LU Update Tolerance** (default value $= 10.0$) are too large. Check also that (sub)program **qphx** has been coded correctly and that all relevant elements of Hx have been assigned their correct values.

ifail $= 7$

An input parameter is invalid.

ifail $= 8$

Numerical error in trying to satisfy the general constraints. The basis is very ill-conditioned.

ifail $= 9$

Not enough integer workspace for the basis factors. Increase **leniz** and rerun e04nk.

ifail $= 10$

Not enough real workspace for the basis factors. Increase **lenz** and rerun e04nk.

ifail $= 11$

The basis is singular after 15 attempts to factorize it (adding slacks where necessary). Either the problem is badly scaled or the value of the optional parameter **LU Factor Tolerance** (default value $= 100.0$) is too large.

ifail = 12

Not enough integer workspace to start solving the problem. Increase **leniz** to at least **miniz** and rerun e04nk.

ifail = 13

Not enough real workspace to start solving the problem. Increase **lenz** to at least **minz** and rerun e04nk.

7 Accuracy

e04nk implements a numerically stable active-set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

8 Further Comments

This section contains a description of the printed output.

8.1 Description of the Printed Output

The following line of summary output ($<$ 80 characters) is produced at every iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn` is the iteration count.

`Step` is the step taken along the computed search direction.

`Ninf` is the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If x is not feasible, `Sinf` gives the sum of the magnitudes of constraint violations. If x is feasible, `Objective` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase, the value of the objective function will be nonincreasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists.

`Norm rg` is $\|d_S\|$, the Euclidean norm of the reduced gradient (see Section 10.3). During the optimality phase, this norm will be approximately zero after a unit step. For FP and LP problems, `Norm rg` is not printed.

The final printout includes a listing of the status of every variable and constraint.

The following describes the printout for each variable. A full stop (.) is printed for any numerical value that is zero.

`Variable` gives the name of the variable. If **nname** $= 1$, a default name is assigned to the jth variable for $j = 1, 2, \ldots, n$. If **nname** $=$ **n** $+$ **m**, the name supplied in **crname**(j) is assigned to the jth variable.

`State` gives the state of the variable (`LL` if nonbasic on its lower bound, `UL` if nonbasic on its upper bound, `EQ` if nonbasic and fixed, `FR` if nonbasic and strictly between its bounds, `BS` if basic and `SBS` if superbasic).

A key is sometimes printed before `State`. Note that unless the optional parameter **Scale Option** $= 0$ (default value $= 2$) is specified, the tests for assigning a key are applied to the variables of the scaled problem.

`A` *Alternative optimum possible*. The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers *might* also change.

`D` *Degenerate*. The variable is basic or superbasic, but it is equal (or very close) to one of its bounds.

`I` *Infeasible*. The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the **Feasibility Tolerance**.

`N` *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Optimality Tolerance**, the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

`Value` is the value of the variable at the final iteration.

`Lower Bound` is the lower bound specified for the variable. `None` indicates that $\mathbf{bl}(j) \leq -bigbnd$.

`Upper Bound` is the upper bound specified for the variable. `None` indicates that $\mathbf{bu}(j) \geq bigbnd$.

`Lagr Mult` is the Lagrange multiplier for the associated bound. This will be zero if `State` is `FR`. If x is optimal, the multiplier should be nonnegative if `State` is `LL`, nonpositive if `State` is `UL` and zero if `State` is `BS` or `SBS`.

`Residual` is the difference between the variable `Value` and the nearer of its (finite) bounds $\mathbf{bl}(j)$ and $\mathbf{bu}(j)$. A blank entry indicates that the associated variable is not bounded (i.e., $\mathbf{bl}(j) \leq -bigbnd$ and $\mathbf{bu}(j) \geq bigbnd$).

The meaning of the printout for linear constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', n replaced by m, $\mathbf{crname}(j)$ replaced by $\mathbf{crname}(n+j)$, $\mathbf{bl}(j)$ and $\mathbf{bu}(j)$ are replaced by $\mathbf{bl}(n+j)$ and $\mathbf{bu}(n+j)$ respectively, and with the following change in the heading:

`Constrnt` gives the name of the linear constraint.

Note that movement off a constraint (as opposed to a variable moving away from its bound) can be interpreted as allowing the entry in the `Residual` column to become positive.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

9 Example

```
e04nk_qphx.m

function [hx, user] = qphx(nstate, ncolh, x, user)
  hx = zeros(ncolh, 1);

  hx(1) = 2*x(1);
  hx(2) = 2*x(2);
  hx(3) = 2*(x(3)+x(4));
  hx(4) = hx(3);
  hx(5) = 2*x(5);
  hx(6) = 2*(x(6)+x(7));
  hx(7) = hx(6);
```

```
n = int32(7);
m = int32(8);
iobj = int32(8);
ncolh = int32(7);
```

```
a = [0.02;
     0.02;
     0.03;
     1;
     0.7;
     0.02;
     0.15;
     -200;
     0.06;
     0.75;
     0.03;
     0.04;
     0.05;
     0.04;
     1;
     -2000;
     0.02;
     1;
     0.01;
     0.08;
     0.08;
     0.8;
     -2000;
     1;
     0.12;
     0.02;
     0.02;
     0.75;
     0.04;
     -2000;
     0.01;
     0.8;
     0.02;
     1;
     0.02;
     0.06;
     0.02;
     -2000;
     1;
     0.01;
     0.01;
     0.97;
     0.01;
     400;
     0.97;
     0.03;
     1;
     400];
ha = [int32(7);
     int32(5);
     int32(3);
     int32(1);
     int32(6);
     int32(4);
     int32(2);
     int32(8);
     int32(7);
     int32(6);
     int32(5);
     int32(4);
     int32(3);
     int32(2);
     int32(1);
     int32(8);
     int32(2);
     int32(1);
     int32(4);
     int32(3);
     int32(7);
     int32(6);
```

```
     int32(8);
     int32(1);
     int32(7);
     int32(3);
     int32(4);
     int32(6);
     int32(2);
     int32(8);
     int32(5);
     int32(6);
     int32(7);
     int32(1);
     int32(2);
     int32(3);
     int32(4);
     int32(8);
     int32(1);
     int32(2);
     int32(3);
     int32(6);
     int32(7);
     int32(8);
     int32(7);
     int32(2);
     int32(1);
     int32(8)];
ka = [int32(1);
     int32(9);
     int32(17);
     int32(24);
     int32(31);
     int32(39);
     int32(45);
     int32(49)];
bl = [0;
     0;
     400;
     100;
     0;
     0;
     0;
     2000;
     -9.999999999999999e+24;
     -9.999999999999999e+24;
     -9.999999999999999e+24;
     -9.999999999999999e+24;
     1500;
     250;
     -9.999999999999999e+24];
bu = [200;
     2500;
     800;
     700;
     1500;
     9.999999999999999e+24;
     9.999999999999999e+24;
     2000;
     60;
     100;
     40;
     30;
     9.999999999999999e+24;
     300;
     9.999999999999999e+24];
start = 'C';
names = {'        '; '        '; '        '; '        '; '        '};
crname = {'...X1...'; '...X2...'; '...X3...'; '...X4...'; '...X5...';
'...X6...'; '...X7...'; '..ROW1..'; '..ROW2..'; '..ROW3..'; '..ROW4..';
'..ROW5..'; '..ROW6..'; '..ROW7..'; '..COST..'};
ns = int32(-1232765364);
```

```
xs = [0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0];
istate = zeros(15, 1, 'int32');
leniz = int32(10000);
lenz = int32(10000);
[cwsav,lwsav,iwsav,rwsav,ifail] = e04wb('e04nk');
[nsOut, xsOut, istateOut, miniz, minz, ninf, sinf, obj, clamda, user,
lwsavOut, iwsavOut, rwsavOut, ifail] = ...
  e04nk(n, m, iobj, ncolh, 'e04nk_qphx', a, ha, ka, bl, bu, start, ...
    names, crname, ns, xs, istate, leniz, lenz, ...
    lwsav, iwsav, rwsav);
 obj
```

```
obj =
  -1.8478e+06
```

Note: *the remainder of this document is intended for more advanced users. Section 10 contains a detailed description of the algorithm which may be needed in order to understand Sections 11 and 12. Section 11 describes the optional parameters which may be set by calls to e04nm. Section 12 describes the quantities which can be requested to monitor the course of the computation.*

10 Algorithmic Details

This section contains a detailed description of the method used by e04nk.

10.1 Overview

e04nk is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method is similar to that of Gill and Murray 1978, and is described in detail by Gill *et al.* 1991. Here we briefly summarize the main features of the method. Where possible, explicit reference is made to the names of variables that are parameters of the function or appear in the printed output.

The method used has two distinct phases: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same (sub)programs. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities (the printed quantity `Sinf`; see Section 12) to the quadratic objective function (the printed quantity `Objective`; see Section 12).

In general, an iterative process is required to solve a quadratic program. Given an iterate (x, s) in both the original variables x and the slack variables s, a new iterate $(\bar{x}, \bar{s})$ is defined by

$$\begin{pmatrix} \bar{x} \\ \bar{s} \end{pmatrix} = \begin{pmatrix} x \\ s \end{pmatrix} + \alpha p, \tag{2}$$

where the *step length* α is a nonnegative scalar (the printed quantity `Step`; see Section 12), and p is called the *search direction*. (For simplicity, we shall consider a typical iteration and avoid reference to the index of the iteration.) Once an iterate is feasible (i.e., satisfies the constraints), all subsequent iterates remain feasible.

10.2 Definition of the Working Set and Search Direction

At each iterate (x, s), a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the value of the optional parameter **Feasibility Tolerance**). The working set is the current prediction of the constraints that hold with equality at a solution of the LP or QP problem. Let m_W denote the number of constraints in the working set (including bounds), and let W denote the associated m_W by $(n+m)$ *working set matrix* consisting of the m_W gradients of the working set constraints.

The search direction is defined so that constraints in the working set remain *unaltered* for any value of the step length. It follows that p must satisfy the identity

$$Wp = 0. \tag{3}$$

This characterization allows p to be computed using any n by n_Z full-rank matrix Z that spans the null space of W. (Thus, $n_Z = n - m_W$ and $WZ = 0$.) The null space matrix Z is defined from a sparse LU factorization of part of W (see (6) and (7)). The direction p will satisfy (3) if

$$p = Zp_Z, \tag{4}$$

where p_Z is any n_Z-vector.

The working set contains the constraints $Ax - s = 0$ and a subset of the upper and lower bounds on the variables (x, s). Since the gradient of a bound constraint $x_j \geq l_j$ or $x_j \leq u_j$ is a vector of all zeros except for ± 1 in position j, it follows that the working set matrix contains the rows of $(A \quad -I)$ and the unit rows associated with the upper and lower bounds in the working set.

The working set matrix W can be represented in terms of a certain column partition of the matrix $(A \quad -I)$ by (conceptually) partitioning the constraints $Ax - s = 0$ so that

$$Bx_B + Sx_S + Nx_N = 0, \tag{5}$$

where B is a square nonsingular basis and x_B, x_S and x_N are the basic, superbasic and nonbasic variables respectively. The nonbasic variables are equal to their upper or lower bounds at (x, s), and the superbasic variables are independent variables that are chosen to improve the value of the current objective function. The number of superbasic variables is n_S (the printed quantity `Ns`; see Section 12). Given values of x_N and x_S, the basic variables x_B are adjusted so that (x, s) satisfies (5).

If P is a permutation matrix such that $(A \quad -I)P = (B \quad S \quad N)$, then W satisfies

$$WP = \begin{pmatrix} B & S & N \\ 0 & 0 & I_N \end{pmatrix}, \tag{6}$$

where I_N is the identity matrix with the same number of columns as N.

The null space matrix Z is defined from a sparse LU factorization of part of W. In particular, Z is maintained in 'reduced gradient' form, using the LUSOL package (see Gill *et al.* 1991) to maintain sparse LU factors of the basis matrix B that alters as the working set W changes. Given the permutation P, the null space basis is given by

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}. \tag{7}$$

This matrix is used only as an operator, i.e., it is never computed explicitly. Products of the form Zv and $Z^{\mathrm{T}}g$ are obtained by solving with B or B^{T}. This choice of Z implies that n_Z, the number of 'degrees of freedom' at (x, s), is the same as n_S, the number of superbasic variables.

Let g_Z and H_Z denote the *reduced gradient* and *reduced Hessian* of the objective function:

$$g_Z = Z^{\mathrm{T}}g \qquad \text{and} \qquad H_Z = Z^{\mathrm{T}}HZ, \tag{8}$$

where g is the objective gradient at (x, s). Roughly speaking, g_Z and H_Z describe the first and second derivatives of an n_S-dimensional *unconstrained* problem for the calculation of p_Z. (The condition estimator of H_Z is the quantity `Cond Hz` in the monitoring file output; see Section 12.)

At each iteration, an upper triangular factor R is available such that $H_Z = R^{\mathrm{T}}R$. Normally, R is computed from $R^{\mathrm{T}}R = Z^{\mathrm{T}}HZ$ at the start of the optimality phase and then updated as the QP working set changes. For efficiency, the dimension of R should not be excessive (say, $n_S \leq 1000$). This is guaranteed if the number of nonlinear variables is 'moderate'.

If the QP problem contains linear variables, H is positive semi-definite and R may be singular with at least one zero diagonal element. However, an inertia-controlling strategy is used to ensure that only the last diagonal element of R can be zero. (See Gill *et al.* 1991 for a discussion of a similar strategy for indefinite quadratic programming.)

If the initial R is singular, enough variables are fixed at their current value to give a nonsingular R. This is equivalent to including temporary bound constraints in the working set. Thereafter, R can become singular only when a constraint is deleted from the working set (in which case no further constraints are deleted until R becomes nonsingular).

10.3 Main Iteration

If the reduced gradient is zero, (x, s) is a constrained stationary point on the working set. During the feasibility phase, the reduced gradient will usually be zero only at a vertex (although it may be zero elsewhere in the presence of constraint dependencies). During the optimality phase, a zero reduced gradient implies that x minimizes the quadratic objective function when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ are defined from the equations

$$W^{\mathrm{T}}\lambda = g(x). \tag{9}$$

A Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\sigma$ when the associated constraint is at its *lower bound*, where σ depends on the value of the optional parameter **Optimality Tolerance**. If a multiplier is nonoptimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by continuing the minimization with the corresponding constraint excluded from the working set. (This step is sometimes referred to as 'deleting' a constraint from the working set.) If optimal multipliers occur during the feasibility phase but the sum of infeasibilities is nonzero, there is no feasible point and the function terminates immediately with **ifail** $= 3$ (see Section 6).

The special form (6) of the working set allows the multiplier vector λ, the solution of (9), to be written in terms of the vector

$$d = \begin{pmatrix} g \\ 0 \end{pmatrix} - \begin{pmatrix} A & -I \end{pmatrix}^{\mathrm{T}}\pi = \begin{pmatrix} g - A^{\mathrm{T}}\pi \\ \pi \end{pmatrix}, \tag{10}$$

where π satisfies the equations $B^{\mathrm{T}}\pi = g_B$, and g_B denotes the basic elements of g. The elements of π are the Lagrange multipliers λ_j associated with the equality constraints $Ax - s = 0$. The vector d_N of nonbasic elements of d consists of the Lagrange multipliers λ_j associated with the upper and lower bound constraints in the working set. The vector d_S of superbasic elements of d is the reduced gradient g_Z in (8). The vector d_B of basic elements of d is zero, by construction. (The Euclidean norm of d_S and the final values of d_S, g and π are the quantities `Norm rg`, `Reduced Gradnt`, `Obj Gradient` and `Dual Activity` in the monitoring file output; see Section 12.)

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the search direction is given by $p = Zp_Z$ (see (7) and (11)). The step length is chosen to maintain feasibility with respect to the satisfied constraints.

There are two possible choices for p_Z, depending on whether or not H_Z is singular. If H_Z is nonsingular, R is nonsingular and p_Z in (4) is computed from the equations

$$R^{\mathrm{T}}Rp_Z = -g_Z, \tag{11}$$

where g_Z is the reduced gradient at x. In this case, $(x, s) + p$ is the minimizer of the objective function subject to the working set constraints being treated as equalities. If $(x, s) + p$ is feasible, α is defined to be unity. In this case, the reduced gradient at $(\bar{x}, \bar{s})$ will be zero, and Lagrange multipliers are computed at the next iteration. Otherwise, α is set to $\alpha_{\mathbf{m}}$, the step to the 'nearest' constraint along p. This constraint is then added to the working set at the next iteration.

If H_Z is singular, then R must also be singular, and an inertia-controlling strategy is used to ensure that only the last diagonal element of R is zero. (See Gill *et al.* 1991 for a discussion of a similar strategy for indefinite quadratic programming.) In this case, p_Z satisfies

$$p_Z^{\mathrm{T}} H_Z p_Z = 0 \qquad \text{and} \qquad g_Z^{\mathrm{T}} p_Z \leq 0, \tag{12}$$

which allows the objective function to be reduced by any step of the form $(x, s) + \alpha p$, where $\alpha > 0$. The vector $p = Z p_Z$ is a direction of unbounded descent for the QP problem in the sense that the QP objective is linear and decreases without bound along p. If no finite step of the form $(x, s) + \alpha p$ (where $\alpha > 0$) reaches a constraint not in the working set, the QP problem is unbounded and the function terminates immediately with **ifail** $= 2$ (see Section 6). Otherwise, α is defined as the maximum feasible step along p and a constraint active at $(x, s) + \alpha p$ is added to the working set for the next iteration.

10.4 Miscellaneous

If the basis matrix is not chosen carefully, the condition of the null space matrix Z in (7) could be arbitrarily high. To guard against this, the function implements a 'basis repair' feature in which the LUSOL package (see Gill *et al.* 1991) is used to compute the rectangular factorization

$$(B \quad S)^{\mathrm{T}} = LU, \tag{13}$$

returning just the permutation P that makes PLP^{T} unit lower triangular. The pivot tolerance is set to require $\left|PLP^{\mathrm{T}}\right|_{ij} \leq 2$, and the permutation is used to define P in (6). It can be shown that $\|Z\|$ is likely to be little more than unity. Hence, Z should be well-conditioned *regardless of the condition of W*. This feature is applied at the beginning of the optimality phase if a potential $B - S$ ordering is known.

The EXPAND procedure (see Gill *et al.* 1989) is used to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. Although there is no absolute guarantee that cycling will not occur, the probability of cycling is extremely small (see Hall and McKinnon 1996). The main feature of EXPAND is that the feasibility tolerance is increased at the start of every iteration. This allows a positive step to be taken at every iteration, perhaps at the expense of violating the bounds on (x, s) by a small amount.

Suppose that the value of the optional parameter **Feasibility Tolerance** is δ. Over a period of K iterations (where K is the value of the optional parameter **Expand Frequency**), the feasibility tolerance actually used by the function (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/K$).

At certain stages the following 'resetting procedure' is used to remove small constraint infeasibilities. First, all nonbasic variables are moved exactly onto their bounds. A count is kept of the number of nontrivial adjustments made. If the count is nonzero, the basic variables are recomputed. Finally, the working feasibility tolerance is reinitialized to 0.5δ.

If a problem requires more than K iterations, the resetting procedure is invoked and a new cycle of iterations is started. (The decision to resume the feasibility phase or optimality phase is based on comparing any constraint infeasibilities with δ.)

The resetting procedure is also invoked when the function reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any nontrivial adjustments are made, iterations are continued.

The EXPAND procedure not only allows a positive step to be taken at every iteration, but also provides a potential *choice* of constraints to be added to the working set. All constraints at a distance α (where $\alpha \leq \alpha_{\mathbf{m}}$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the largest angle with the search direction is added to the working set. This strategy helps keep the basis matrix B well-conditioned.

11 Optional Parameters

Several optional parameters in e04nk define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of e04nk these optional parameters have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you wish to use the default values for *all* optional parameters. A complete list of optional parameters and their default values is given in Section 11.1.

Optional parameters may be specified by calling e04nm before a call to e04nk.

e04nm can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
[lwsav, iwsav, rwsav, inform] = e04nm('Print Level = 5', lwsav, iwsav,
rwsav);
```

e04nm should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by you are set to their default values. Optional parameters specified by you are unaltered by e04nk (unless they define invalid values) and so remain in effect for subsequent calls unless altered by you.

11.1 Optional Parameter Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 11.2. The minimum abbreviation of each keyword is underlined. The qualifier may be omitted. The letters i and r denote integer and double values required with certain options. The default value of an option is used whenever the condition $|i| \geq 100000000$ is satisfied. The number ϵ is a generic notation for ***machine precision*** (see x02aj).

Optional Parameters	**Default Values**
Check Frequency	Default $= 60$
Crash Option	Default $= 2$
Crash Tolerance	Default $= 0.1$
Defaults	
Expand Frequency	Default $= 10000$
Factorization Frequency	Default $= 100$
Feasibility Tolerance	Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$
Infinite Bound Size	Default $= 10^{20}$
Infinite Step Size	Default $= \max\left(bigbnd, 10^{20}\right)$
Iteration Limit	Default $= \max(50, 5(n+m))$
Iters	See **Iteration Limit**
Itns	See **Iteration Limit**
LU Factor Tolerance	Default $= 100.0$
LU Singularity Tolerance	Default $= \epsilon^{0.67}$
LU Update Tolerance	Default $= 10.0$. See **LU Factor Tolerance**.
List	Default for $e04nk =$ **List**
Maximize	See **Minimize**
Minimize	Default
Monitoring File	Default $= -1$
Nolist	Default for $e04nk =$ **Nolist**. See **List**.
Optimality Tolerance	Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$
Partial Price	Default $= 10$
Pivot Tolerance	Default $= \epsilon^{0.67}$

Print Level	Default for e04nk $= 10$
	Default for e04nk $= 0$
Rank Tolerance	Default $= 100\epsilon$
Scale Option	Default $= 2$
Scale Tolerance	Default $= 0.9$
Superbasics Limit	Default $= \min(n_H + 1, n)$

11.2 Description of the Optional Parameters

Check Frequency i Default $= 60$

Every ith iteration after the most recent basis factorization, a numerical test is made to see if the current solution (x, s) satisfies the linear constraints $Ax - s = 0$. If the largest element of the residual vector $r = Ax - s$ is judged to be too large, the current basis is refactorized and the basic variables recomputed to satisfy the constraints more accurately. If $i < 0$, the default value is used. If $i = 0$, the value $i = 99999999$ is used and effectively no checks are made.

Crash Option i Default $= 2$

Note that this option does not apply when **start** = 'W' (see Section 5).

If **start** = 'C', an internal Crash procedure is used to select an initial basis from various rows and columns of the constraint matrix $\begin{pmatrix} A & -I \end{pmatrix}$. The value of i determines which rows and columns are initially eligible for the basis, and how many times the Crash procedure is called. If $i = 0$, the all-slack basis $B = -I$ is chosen. If $i = 1$, the Crash procedure is called once (looking for a triangular basis in all rows and columns of the linear constraint matrix A). If $i = 2$, the Crash procedure is called twice (looking at any *equality* constraints first followed by any *inequality* constraints). If $i < 0$ or $i > 2$, the default value is used.

If $i = 1$ or 2, certain slacks on inequality rows are selected for the basis first. (If $i = 2$, numerical values are used to exclude slacks that are close to a bound.) The Crash procedure then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to 'pivot' on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

Crash Tolerance r Default $= 0.1$

This value allows the Crash procedure to ignore certain 'small' nonzero elements in the constraint matrix A while searching for a triangular basis. For each column of A, if $a_{\max}$ is the largest element in the column, other nonzeros in that column are ignored if they are less than (or equal to) $a_{\max} \times r$.

When $r > 0$, the basis obtained by the Crash procedure may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis with more column variables and fewer (arbitrary) slacks. A feasible solution may be reached earlier for some problems. If $r < 0$ or $r \geq 1$, the default value is used.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Expand Frequency i Default $= 10000$

This option is part of an anti-cycling procedure (see Section 10.4) designed to allow progress even on highly degenerate problems.

For LP problems, the strategy is to force a positive step at every iteration, at the expense of violating the constraints by a small amount. Suppose that the value of the optional parameter **Feasibility Tolerance** is δ. Over a period of i iterations, the feasibility tolerance actually used by e04nk (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For QP problems, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can only occur when the current solution is at a vertex of the feasible region.)

Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

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

If $i < 0$, the default value is used. If $i = 0$, the value $i = 99999999$ is used and effectively no anti-cycling procedure is invoked.

Factorization Frequency i Default $= 100$

If $i > 0$, at most i basis changes will occur between factorizations of the basis matrix. For LP problems, the basis factors are usually updated at every iteration. For QP problems, fewer basis updates will occur as the solution is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly according to the value of optional parameter **Check Frequency** to ensure that the linear constraints $Ax - s = 0$ are satisfied. If necessary, the basis will be refactorized before the limit of i updates is reached. If $i \leq 0$, the default value is used.

Feasibility Tolerance r Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$

If $r \geq \epsilon$, r defines the maximum acceptable *absolute* violation in each constraint at a 'feasible' point (including slack variables). For example, if the variables and the coefficients in the linear constraints are of order unity, and the latter are correct to about five decimal digits, it would be appropriate to specify r as 10^{-5}. If $r < \epsilon$, the default value is used.

e04nk attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is assumed to be *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. Note that the function does *not* attempt to find the minimum value of `Sinf`.

If the constraints and variables have been scaled (see **Scale Option**), then feasibility is defined in terms of the scaled problem (since it is more likely to be meaningful).

Infinite Bound Size r Default $= 10^{20}$

If $r > 0$, r defines the 'infinite' bound $bigbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $bigbnd$ will be regarded as plus infinity (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as minus infinity). If $r \leq 0$, the default value is used.

Infinite Step Size r Default $= \max\left(bigbnd, 10^{20}\right)$

If $r > 0$, r specifies the magnitude of the change in variables that will be considered a step to an unbounded solution. (Note that an unbounded solution can occur only when the Hessian is not positive-definite.) If the change in x during an iteration would exceed the value of r, the objective function is considered to be unbounded below in the feasible region. If $r \leq 0$, the default value is used.

Iteration Limit i Default $= \max(50, 5(n + m))$
Iters
Itns

The value of i specifies the maximum number of iterations allowed before termination. Setting $i = 0$ and **Print Level** > 0 means that the workspace needed to start solving the problem will be computed and printed, but no iterations will be performed. If $i < 0$, the default value is used.

List Default for $e04nk =$ **List**
Nolist Default for $e04nk =$ **Nolist**

Normally each optional parameter specification is printed as it is supplied. Optional parameter **Nolist** may be used to suppress the printing and optional parameter **List** may be used to restore printing.

LU Factor Tolerance r_1 Default $= 100.0$
LU Update Tolerance r_2 Default $= 10.0$

The values of r_1 and r_2 affect the stability and sparsity of the basis factorization $B = LU$, during refactorization and updates respectively. The lower triangular matrix L is a product of matrices of the form

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

where the multipliers μ will satisfy $|\mu| \leq r_i$. The default values of r_1 and r_2 usually strike a good compromise between stability and sparsity. For large and relatively dense problems, setting r_1 and r_2 to 25 (say) may give a marked improvement in sparsity without impairing stability to a serious degree.

Note that for band matrices it may be necessary to set r_1 in the range $1 \leq r_1 < 2$ in order to achieve stability. If $r_1 < 1$ or $r_2 < 1$, the default value is used.

LU Singularity Tolerance r Default $= \epsilon^{0.67}$

If $r > 0$, r defines the singularity tolerance used to guard against ill-conditioned basis matrices. Whenever the basis is refactorized, the diagonal elements of U are tested as follows. If $|u_{jj}| \leq r$ or $|u_{jj}| < r \times \max_i |u_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. If $r \leq 0$, the default value is used.

Minimize Default
Maximize

This option specifies the required direction of the optimization. It applies to both linear and nonlinear terms (if any) in the objective function. Note that if two problems are the same except that one minimizes $f(x)$ 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 (see Section 10.3) will be reversed.

Monitoring File i Default $= -1$

If $i \geq 0$ and **Print Level** > 0 (see **Print Level**), monitoring information produced by e04nk is sent to a file with logical unit number i. If $i < 0$ and/or **Print Level** $= 0$, the default value is used and hence no monitoring information is produced.

Optimality Tolerance r Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$

If $r \geq \epsilon$, r is used to judge the size of the reduced gradients $d_j = g_j - \pi^{\mathrm{T}} a_j$. By definition, the reduced gradients for basic variables are always zero. Optimality is declared if the reduced gradients for any nonbasic variables at their lower or upper bounds satisfy $-r \times \max(1, \|\pi\|) \leq d_j \leq r \times \max(1, \|\pi\|)$, and if $|d_j| \leq r \times \max(1, \|\pi\|)$ for any superbasic variables. If $r < \epsilon$, the default value is used.

Partial Price i Default $= 10$

Note that this option does not apply to QP problems.

This option is recommended for large FP or LP problems that have significantly more variables than constraints (i.e., $n \gg m$). It reduces the work required for each pricing operation (i.e., when a nonbasic variable is selected to enter the basis). If $i = 1$, all columns of the constraint matrix $(A \quad -I)$ are searched. If $i > 1$, A and $-I$ are partitioned to give i roughly equal segments A_j, K_j, for $j = 1, 2, \ldots, p$ (modulo p). If the previous pricing search was successful on A_{j-1}, K_{j-1}, the next search begins on the segments A_j, K_j. 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 enter the basis. If nothing is found, the search continues on the next segments A_{j+1}, K_{j+1}, and so on. If $i \leq 0$, the default value is used.

Pivot Tolerance r Default $= \epsilon^{0.67}$

If $r > 0$, r is used to prevent columns entering the basis if they would cause the basis to become almost singular. If $r \leq 0$, the default value is used.

Print Level i Default for e04nk $= 10$
Default for e04nk $= 0$

The value of i controls the amount of printout produced by e04nk, as indicated below. A detailed description of the printed output is given in Section 8.1 (summary output at each iteration and the final solution) and Section 12 (monitoring information at each iteration). Note that the summary output will not exceed 80 characters per line and that the monitoring information will not exceed 120 characters per line. If $i < 0$, the default value is used.

The following printout is sent to the current advisory message unit (as defined by x04ab):

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output for each iteration (no printout of the final solution).
≥ 10	The final solution and one line of summary output for each iteration.

The following printout is sent to the logical unit number defined by the optional parameter **Monitoring File**:

i	**Output**
0	No output.
1	The final solution only.
5	One long line of output for each iteration (no printout of the final solution).
≥ 10	The final solution and one long line of output for each iteration.
≥ 20	The final solution, one long line of output for each iteration, matrix statistics (initial status of rows and columns, number of elements, density, biggest and smallest elements, etc.), details of the scale factors resulting from the scaling procedure (if **Scale Option** $= 1$ or 2 (see the description of the optional parameter **Scale Option**), basis factorization statistics and details of the initial basis resulting from the Crash procedure (if **start** = 'C'; see Section 5).

If **Print Level** > 0 and the unit number defined by optional parameter **Monitoring File** is the same as that defined by x04ab, then the summary output is suppressed.

Rank Tolerance r Default $= 100\epsilon$

Scale Option i Default $= 2$

This option enables you to scale the variables and constraints using an iterative procedure due to Fourer 1982, which attempts to compute row scales r_i and column scales c_j such that the scaled matrix coefficients $\bar{a}_{ij} = a_{ij} \times \left(c_j/r_i\right)$ are as close as possible to unity. This may improve the overall efficiency on some problems. (The lower and upper bounds on the variables and slacks for the scaled problem are redefined as $\bar{l}_j = l_j/c_j$ and $\bar{u}_j = u_j/c_j$ respectively, where $c_j \equiv r_{j-n}$ if $j > n$.)

If $i = 0$, no scaling is performed. If $i = 1$, all rows and columns of the constraint matrix A are scaled. If $i = 2$, an additional scaling is performed that may be helpful when the solution x is large; it takes into account columns of $\left(A \quad -I \right)$ that are fixed or have positive lower bounds or negative upper bounds. If $i < 0$ or $i > 2$, the default value is used.

Scale Tolerance r Default $= 0.9$

Note that this option does not apply when **Scale Option** $= 0$.

If $0 < r < 1$, r is used to control the number of scaling passes to be made through the constraint matrix A. At least 3 (and at most 10) passes will be made. More precisely, let a_p denote the largest column ratio (i.e., $\frac{\text{'biggest' element}}{\text{'smallest' element}}$ in some sense) after the pth scaling pass through A. The scaling procedure is terminated if $a_p \geq a_{p-1} \times r$ for some $p \geq 3$. Thus, increasing the value of r from 0.9 to 0.99 (say) will probably increase the number of passes through A. If $r \leq 0$ or $r \geq 1$, the default value is used.

Superbasics Limit i Default $= \min(n_H + 1, n)$

Note that this option does not apply to FP or LP problems.

The value of i specifies 'how nonlinear' you expect the QP problem to be. If $i \leq 0$, the default value is used.

12 Description of Monitoring Information

This section describes the intermediate printout and final printout which constitutes the monitoring information produced by e04nk. (See also the description of the optional parameters **Monitoring File** and **Print Level**.) You can control the level of printed output.

When **Print Level** $= 5$ or ≥ 10 and **Monitoring File** ≥ 0, the following line of intermediate printout ($<$ 120 characters) is produced at every iteration on the unit number specified by optional parameter **Monitoring File**. Unless stated otherwise, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn` is the iteration count.

`pp` is the partial price indicator. The variable selected by the last pricing operation came from the ppth partition of A and $-I$. Note that pp is reset to zero whenever the basis is refactorized.

`dj` is the value of the reduced gradient (or reduced cost) for the variable selected by the pricing operation at the start of the current iteration.

`+S` is the variable selected by the pricing operation to be added to the superbasic set.

`-S` is the variable chosen to leave the superbasic set.

`-BS` is the variable removed from the basis (if any) to become nonbasic.

`Step` is the value of the step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. If a variable is made superbasic during the current iteration (i.e., `+S` is positive), `Step` will be the step to the nearest bound. During the optimality phase, the step can be greater than unity only if the reduced Hessian is not positive-definite.

`Pivot` is the rth element of a vector y satisfying $By = a_q$ whenever a_q (the qth column of the constraint matrix $(A \quad -I)$) replaces the rth column of the basis matrix B. Wherever possible, `Step` is chosen so as to avoid extremely small values of `Pivot` (since they may cause the basis to be nearly singular). In extreme cases, it may be necessary to increase the value of the optional parameter **Pivot Tolerance** to exclude very small elements of y from consideration during the computation of `Step`.

`Ninf` is the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If x is not feasible, `Sinf` gives the sum of the magnitudes of constraint violations. If x is feasible, `Objective` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase, the value of the objective function will be nonincreasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists.

`L` is the number of nonzeros in the basis factor L. Immediately after a basis factorization $B = LU$, this entry contains `lenL`. Further nonzeros are added to `L` when various columns of B are later replaced. (Thus, `L` increases monotonically.)

`U` is the number of nonzeros in the basis factor U. Immediately after a basis factorization $B = LU$, this entry contains `lenU`. 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` is the number of compressions required to recover workspace 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 it does not, increase **leniz** and **lenz** by at least $\mathtt{L} + \mathtt{U}$ and rerun e04nk (possibly using **start** = 'W'; see Section 5).

`Norm rg` is $\|d_S\|$, the Euclidean norm of the reduced gradient (see Section 10.3). During the optimality phase, this norm will be approximately zero after a unit step. For FP and LP problems, `Norm rg` is not printed.

`Ns` is the current number of superbasic variables. For FP and LP problems, `Ns` is not printed.

`Cond Hz` is a lower bound on the condition number of the reduced Hessian (see Section 10.2). The larger this number, the more difficult the problem. For FP and LP
problems, `Cond Hz` is not printed.

When **Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by optional parameter **Monitoring File** whenever the matrix B or $B_S = (B \quad S)^{\mathrm{T}}$ is factorized. Gaussian elimination is used to compute an LU factorization of B or B_S, where PLP^{T} is a lower triangular matrix and PUQ is an upper triangular matrix for some permutation matrices P and Q. The factorization is stabilized in the manner described under the optional parameter **LU Factor Tolerance** (default value = 100.0).

`Factorize` is the factorization count.

`Demand` is a code giving the reason for the present factorization as follows:

Code	**Meaning**
0	First LU factorization.
1	The number of updates reached the value of the optional parameter **Factorization Frequency**.
2	The number of nonzeros in the updated factors has increased significantly.
7	Not enough storage to update factors.
10	Row residuals too large (see the description for the optional parameter **Check Frequency**).
11	Ill-conditioning has caused inconsistent results.

`Iteration` is the iteration count.

`Nonlinear` is the number of nonlinear variables in the current basis B (not printed if B_S is factorized).

`Linear` is the number of linear variables in B (not printed if B_S is factorized).

`Slacks` is the number of slack variables in B (not printed if B_S is factorized).

`Elems`	is the number of nonzeros in B (not printed if B_S is factorized).
`Density`	is the percentage nonzero density of B (not printed if B_S is factorized). More precisely, $\texttt{Density} = 100 \times \texttt{Elems}/(\texttt{Nonlinear} + \texttt{Linear} + \texttt{Slacks})^2$.
`Compressns`	is the number of times the data structure holding the partially factorized matrix needed to be compressed, in order to recover unused workspace. Ideally, it should be zero. If it is more than 3 or 4, increase **leniz** and **lenz** and rerun e04nk (possibly using **start** = 'W'; see Section 5).
`Merit`	is 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`	is the number of nonzeros in L.
`lenU`	is the number of nonzeros in U.
`Increase`	is the percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B. More precisely, $\texttt{Increase} = 100 \times (\texttt{lenL} + \texttt{lenU} - \texttt{Elems})/\texttt{Elems}$.
`m`	is the number of rows in the problem. Note that $\texttt{m} = \texttt{Ut} + \texttt{Lt} + \texttt{bp}$.
`Ut`	is the number of triangular rows of B at the top of U.
`d1`	is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.
`Lmax`	is the maximum subdiagonal element in the columns of L. This will not exceed the value of the optional parameter **LU Factor Tolerance**.
`Bmax`	is the maximum nonzero element in B (not printed if B_S is factorized).
`BSmax`	is the maximum nonzero element in B_S (not printed if B is factorized).
`Umax`	is the maximum nonzero element in U, excluding elements of B that remain in U unchanged. (For example, if a slack variable is in the basis, the corresponding row of B will become a row of U without modification. Elements in such rows will not contribute to `Umax`. If the basis is strictly triangular then *none* of the elements of B will contribute and `Umax` will be zero.) Ideally, `Umax` should not be significantly larger than `Bmax`. If it is several orders of magnitude larger, it may be advisable to reset the optional parameter **LU Factor Tolerance** to some value nearer unity. `Umax` is not printed if B_S is factorized.
`Umin`	is the magnitude of the smallest diagonal element of PUQ (not printed if B_S is factorized).
`Growth`	is the value of the ratio `Umax`/`Bmax`, which should not be too large. Providing `Lmax` is not large (say, < 10.0), the ratio $\max(\texttt{Bmax}, \texttt{Umax})/\texttt{Umin}$ 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 and the modified basis is refactorized.) `Growth` is not printed if B_S is factorized.
`Lt`	is the number of triangular columns of B at the left of L.

`bp`	is the size of the 'bump' or block to be factorized nontrivially after the triangular rows and columns of B have been removed.
`d2`	is the number of columns remaining when the density of the basis matrix being factorized has reached 0.6.

When **Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following lines of intermediate printout (< 80 characters) are produced on the unit number specified by optional parameter **Monitoring File** whenever **start** = 'C' (see Section 5). They refer to the number of columns selected by the Crash procedure during each of several passes through A, whilst searching for a triangular basis matrix.

`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., **istate**$(j) = 3$ for some $j \leq n$). It will be a subset of the columns for which **istate**$(j) = 3$ was specified.
`Unit`	is the number of unit columns in the basis.
`Double`	is the number of double columns in the basis.
`Triangle`	is the number of triangular columns in the basis.
`Pad`	is the number of slacks used to pad the basis (to make it a nonsingular triangle).

When **Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following lines of intermediate printout (< 80 characters) are produced on the unit number specified by optional parameter **Monitoring File**. They refer to the elements of the **names** array (see Section 5).

`Name`	gives the name for the problem (blank if problem unnamed).
`Status`	gives the exit status for the problem (i.e., `Optimal soln`, `Weak soln`, `Unbounded`, `Infeasible`, `Excess itns`, `Error condn` or `Feasble soln`) followed by details of the direction of the optimization (i.e., `(Min)` or `(Max)`).
`Objective`	gives the name of the free row for the problem (blank if objective unnamed).
`RHS`	gives the name of the constraint right-hand side for the problem (blank if objective unnamed).
`Ranges`	gives the name of the ranges for the problem (blank if objective unnamed).
`Bounds`	gives the name of the bounds for the problem (blank if objective unnamed).

When **Print Level** $= 1$ or ≥ 10 and **Monitoring File** ≥ 0, the following lines of final printout (< 120 characters) are produced on the unit number specified by optional parameter **Monitoring File**.

Let a_j denote the jth column of A, for $j = 1, 2, \ldots, n$. The following describes the printout for each column (or variable). A full stop (.) is printed for any numerical value that is zero.

`Number`	is the column number j. (This is used internally to refer to x_j in the intermediate output.)
`Column`	gives the name of x_j.
`State`	gives the state of the variable (`LL` if nonbasic on its lower bound, `UL` if nonbasic on its upper bound, `EQ` if nonbasic and fixed, `FR` if nonbasic and strictly between its bounds, `BS` if basic and `SBS` if superbasic). A key is sometimes printed before `State`. Note that unless the optional parameter **Scale Option** $= 0$ is specified, the tests for assigning a key are applied to the variables of the scaled problem.

A *Alternative optimum possible*. The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled D), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers *might* also change.

D *Degenerate*. The variable is basic or superbasic, but it is equal (or very close) to one of its bounds.

I *Infeasible*. The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the **Feasibility Tolerance**.

N *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Optimality Tolerance**, the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

`Activity` is the value of x_j at the final iterate.

`Obj Gradient` is the value of g_j at the final iterate. For FP problems, g_j is set to zero.

`Lower Bound` is the lower bound specified for the variable. `None` indicates that $\mathbf{bl}(j) \leq -bigbnd$.

`Upper Bound` is the upper bound specified for the variable. `None` indicates that $\mathbf{bu}(j) \geq bigbnd$.

`Reduced Gradnt` is the value of d_j at the final iterate (see Section 10.3). For FP problems, d_j is set to zero.

`m + j` is the value of $m + j$.

Let v_i denote the ith row of A, for $i = 1, 2, \ldots, m$. The following describes the printout for each row (or constraint). A full stop (.) is printed for any numerical value that is zero.

`Number` is the value of $n + i$. (This is used internally to refer to s_i in the intermediate output.)

`Row` gives the name of ν_i.

`State` gives the state of v_i (LL if active on its lower bound, UL if active on its upper bound, EQ if active and fixed, BS if inactive when s_i is basic and SBS if inactive when s_i is superbasic).

A key is sometimes printed before `State`. Note that unless the optional parameter **Scale Option** $= 0$ is specified, the tests for assigning a key are applied to the variables of the scaled problem.

A *Alternative optimum possible*. The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled D), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers *might* also change.

D *Degenerate*. The variable is basic or superbasic, but it is equal (or very close) to one of its bounds.

	`I` *Infeasible*. The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the **Feasibility Tolerance**.
	`N` *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Optimality Tolerance**, the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.
`Activity`	is the value of v_i at the final iterate.
`Slack Activity`	is the value by which the row differs from its nearest bound. (For the free row (if any), it is set to `Activity`.)
`Lower Bound`	is the lower bound specified for the variable. `None` indicates that $\mathbf{bl}(j) \leq -bigbnd$.
`Upper Bound`	is the upper bound specified for the variable. `None` indicates that $\mathbf{bu}(j) \geq bigbnd$.
`Dual Activity`	is the value of the dual variable π_i (the Lagrange multiplier for ν_i; see Section 10.3). For FP problems, π_i is set to zero.
`i`	gives the index i of the ith row.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.