NAG Toolbox

nag_opt_nlp1_sparse_solve (e04ug)

1 Purpose

nag_opt_nlp1_sparse_solve (e04ug) solves sparse nonlinear programming problems.

2 Syntax

```
[a, ns, xs, istate, clamda, miniz, minz, ninf, sinf, obj, user, lwsav, iwsav,
rwsav, ifail] = nag_opt_nlp1_sparse_solve(confun, objfun, n, m, ncnln, nonln,
njnln, iobj, a, ha, ka, bl, bu, start, names, ns, xs, istate, clamda, lwsav,
iwsav, rwsav, 'nnz', nnz, 'nname', nname, 'leniz', leniz, 'lenz', lenz, 'user',
user)
```

```
[a, ns, xs, istate, clamda, miniz, minz, ninf, sinf, obj, user, lwsav, iwsav,
rwsav, ifail] = e04ug(confun, objfun, n, m, ncnln, nonln, njnln, iobj, a, ha,
ka, bl, bu, start, names, ns, xs, istate, clamda, lwsav, iwsav, rwsav, 'nnz',
nnz, 'nname', nname, 'leniz', leniz, 'lenz', lenz, 'user', user)
```

Before calling nag_opt_nlp1_sparse_solve (e04ug), or the option setting function nag_opt_nlp1_sparse_option_string (e04uj), nag_opt_init (e04wb) **must** be called.

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: **lenz** and **leniz** were made optional.

3 Description

nag_opt_nlp1_sparse_solve (e04ug) is designed to solve a class of nonlinear programming problems that are assumed to be stated in the following general form:

$$\underset{x\in R^n}{\text{minimize}} f(x) \quad \text{subject to} \quad l \leq \begin{Bmatrix} x \\ F(x) \\ Gx \end{Bmatrix} \leq u, \tag{1}$$

where $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ is a set of variables, $f(x)$ is a smooth scalar objective function, l and u are constant lower and upper bounds, $F(x)$ is a vector of smooth nonlinear constraint functions $\{F_i(x)\}$ and G is a *sparse* matrix.

The constraints involving F and Gx are called the *general constraints*. Note that upper and lower bounds are specified for all variables and constraints. This form allows full generality in specifying various types of constraint. In particular, the jth constraint can be defined as an *equality* by setting $l_j = u_j$. If certain bounds are not present, the associated elements of l or u can be set to special values that will be treated as $-\infty$ or $+\infty$. (See the description of the optional parameter **Infinite Bound Size**.)

nag_opt_nlp1_sparse_solve (e04ug) converts the upper and lower bounds on the m elements of F and Gx 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:

$$\underset{x\in R^n, s\in R^m}{\text{minimize}} f(x) \quad \text{subject to} \quad \{Gx\} - s = 0, \quad l \leq \begin{Bmatrix} x \\ s \end{Bmatrix} \leq u. \tag{2}$$

Since the slack variables s are subject to the same upper and lower bounds as the elements of F and Gx, the bounds on F and Gx can simply be thought of as bounds on the combined vector (x, s). The elements of x and s are partitioned into *basic*, *nonbasic* and *superbasic variables* defined as follows:

– a basic variable (x_j say) is the jth variable associated with the jth column of the basis matrix B;

– a nonbasic variable is a variable that is temporarily fixed at its current value (usually its upper or lower bound);

– a superbasic variable is a nonbasic variable which is not at one of its bounds that is free to move in any desired direction (namely one that will improve the value of the objective function or reduce the sum of infeasibilities).

For example, in the simplex method (see Gill *et al.* (1981)) the elements of x can be partitioned at each vertex into a set of m basic variables (all non-negative) and a set of $(n-m)$ nonbasic variables (all zero). This is equivalent to partitioning the columns of the constraint matrix as $\begin{pmatrix} B & N \end{pmatrix}$, where B contains the m columns that correspond to the basic variables and N contains the $(n-m)$ columns that correspond to the nonbasic variables. Note that B is square and nonsingular.

The optional parameter **Maximize** may be used to specify an alternative problem in which $f(x)$ is maximized. If the objective function is nonlinear and all the constraints are linear, F is absent and the problem is said to be *linearly constrained*. In general, the objective and constraint functions are *structured* in the sense that they are formed from sums of linear and nonlinear functions. This structure can be exploited by the function during the solution process as follows.

Consider the following nonlinear optimization problem with four variables (u, v, z, w):

$$\underset{u,v,z,w}{\text{minimize}} \quad (u+v+z)^2 + 3z + 5w$$

subject to the constraints

$$\begin{array}{rl} u^2+v^2+z & = 2 \\ u^4+v^4 \quad +w & = 4 \\ 2u+4v & \geq 0 \end{array}$$

and to the bounds

$$\begin{array}{l} z \geq 0 \\ w \geq 0. \end{array}$$

This problem has several characteristics that can be exploited by the function:

– the objective function is nonlinear. It is the sum of a *nonlinear* function of the variables (u, v, z) and a *linear* function of the variables (z, w);

– the first two constraints are nonlinear. The third is linear;

– each nonlinear constraint function is the sum of a *nonlinear* function of the variables (u, v) and a *linear* function of the variables (z, w).

The nonlinear terms are defined by **objfun** and **confun** (see Section 5), which involve only the appropriate subset of variables.

For the objective, we define the function $f(u, v, z) = (u+v+z)^2$ to include only the nonlinear part of the objective. The three variables (u, v, z) associated with this function are known as the *nonlinear objective variables*. The number of them is given by **nonln** (see Section 5) and they are the only variables needed in **objfun**. The linear part $3z + 5w$ of the objective is stored in row **iobj** (see Section 5) of the (constraint) Jacobian matrix A (see below).

Thus, if x' and y' denote the nonlinear and linear objective variables, respectively, the objective may be re-written in the form

$$f(x') + c^{\mathrm{T}}x' + d^{\mathrm{T}}y',$$

where $f(x')$ is the nonlinear part of the objective and c and d are constant vectors that form a row of A. In this example, $x' = (u, v, z)$ and $y' = w$.

Similarly for the constraints, we define a vector function $F(u, v)$ to include just the nonlinear terms. In this example, $F_1(u, v) = u^2 + v^2$ and $F_2(u, v) = u^4 + v^4$, where the two variables (u, v) are known as the *nonlinear Jacobian variables*. The number of them is given by **njnln** (see Section 5) and they are

the only variables needed in **confun**. Thus, if x'' and y'' denote the nonlinear and linear Jacobian variables, respectively, the constraint functions and the linear part of the objective have the form

$$\begin{pmatrix} F(x'') + A_2 y'' \\ A_3 x'' + A_4 y'' \end{pmatrix}, \tag{3}$$

where $x'' = (u, v)$ and $y'' = (z, w)$ in this example. This ensures that the Jacobian is of the form

$$A = \begin{pmatrix} J(x'') & A_2 \\ A_3 & A_4 \end{pmatrix},$$

where $J(x'') = \dfrac{\partial F(x'')}{\partial x}$. Note that $J(x'')$ *always* appears in the *top left-hand corner* of A.

The inequalities $l_1 \leq F(x'') + A_2 y'' \leq u_1$ and $l_2 \leq A_3 x'' + A_4 y'' \leq u_2$ implied by the constraint functions in (3) are known as the *nonlinear* and *linear* constraints, respectively. The nonlinear constraint vector $F(x'')$ in (3) and (optionally) its partial derivative matrix $J(x'')$ are set in **confun**. The matrices A_2, A_3 and A_4 contain any (constant) linear terms. Along with the sparsity pattern of $J(x'')$ they are stored in the arrays **a**, **ha** and **ka** (see Section 5).

In general, the vectors x' and x'' have different dimensions, but they *always overlap*, in the sense that the shorter vector is always the beginning of the other. In the above example, the nonlinear Jacobian variables (u, v) are an ordered subset of the nonlinear objective variables (u, v, z). In other cases it could be the other way round (whichever is the most convenient), but the first way keeps $J(x'')$ as small as possible.

Note that the nonlinear objective function $f(x')$ may involve either a subset or superset of the variables appearing in the nonlinear constraint functions $F(x'')$. Thus, **nonln** $\leq$ **njnln** (or vice-versa). Sometimes the objective and constraints really involve *disjoint sets of nonlinear variables*. In such cases the variables should be ordered so that **nonln** $>$ **njnln** and $x' = (x'', x''')$, where the objective is nonlinear in just the last vector x'''. The first **njnln** elements of the gradient array **objgrd** should also be set to zero in **objfun**. This is illustrated in Section 10.

If all elements of the constraint Jacobian are known (i.e., the optional parameter **Derivative Level** $= 2$ or 3), any constant elements may be assigned their correct values in **a**, **ha** and **ka**. The corresponding elements of the constraint Jacobian array **fjac** need not be reset in **confun**. This includes values that are identically zero as constraint Jacobian elements are assumed to be zero unless specified otherwise. It must be emphasized that, if **Derivative Level** $= 0$ or 1, unassigned elements of **fjac** are *not* treated as constant; they are estimated by finite differences, at nontrivial expense.

If there are no nonlinear constraints in (1) and $f(x)$ is linear or quadratic, then it may be more efficient to use nag_opt_qpconvex2_sparse_solve (e04nq) to solve the resulting linear or quadratic programming problem, or one of nag_opt_lp_solve (e04mf), nag_opt_lsq_lincon_solve (e04nc) or nag_opt_qp_dense_solve (e04nf) if G is a *dense* matrix. If the problem is dense and does have nonlinear constraints then one of nag_opt_nlp1_rcomm (e04uf), nag_opt_lsq_gencon_deriv (e04us) or nag_opt_nlp2_solve (e04wd) (as appropriate) should be used instead.

You must supply an initial estimate of the solution to (1), together with versions of **objfun** and **confun** that define $f(x')$ and $F(x'')$, respectively, and as many first partial derivatives as possible. Note that if there are any nonlinear constraints, then the *first* call to **confun** will precede the *first* call to **objfun**.

nag_opt_nlp1_sparse_solve (e04ug) is based on the SNOPT package described in Gill *et al.* (2002), which in turn utilizes functions from the MINOS package (see Murtagh and Saunders (1995)). It incorporates a sequential quadratic programming (SQP) method that obtains search directions from a sequence of quadratic programming (QP) subproblems. Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to a linearization of the constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point. Further details can be found in Section 11.

Throughout this document the symbol ϵ is used to represent the ***machine precision*** (see nag_machine_precision (x02aj)).

4 References

Conn A R (1973) Constrained optimization using a nondifferentiable penalty function *SIAM J. Numer. Anal.* **10** 760–779

Eldersveld S K (1991) Large-scale sequential quadratic programming algorithms *PhD Thesis* Department of Operations Research, Stanford University, Stanford

Fletcher R (1984) An l_1 penalty method for nonlinear constraints *Numerical Optimization 1984* (eds P T Boggs, R H Byrd and R B Schnabel) 26–40 SIAM Philadelphia

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

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 (1986) Users' guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming *Report SOL 86-2* Department of Operations Research, Stanford University

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 (1992) Some theoretical properties of an augmented Lagrangian merit function *Advances in Optimization and Parallel Computing* (ed P M Pardalos) 101–128 North Holland

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

Hock W and Schittkowski K (1981) *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer–Verlag

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

Ortega J M and Rheinboldt W C (1970) *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press

Powell M J D (1974) Introduction to constrained optimization *Numerical Methods for Constrained Optimization* (eds P E Gill and W Murray) 1–28 Academic Press

5 Parameters

5.1 Compulsory Input Parameters

1: **confun** – SUBROUTINE, supplied by the NAG Library or the user.

confun must calculate the vector $F(x)$ of nonlinear constraint functions and (optionally) its Jacobian $\left(=\dfrac{\partial F}{\partial x}\right)$ for a specified n_1'' ($\leq n$) element vector x. If there are no nonlinear constraints (i.e., **ncnln** $= 0$), **confun** will never be called by nag_opt_nlp1_sparse_solve (e04ug) and **confun** may be the string `nag_opt_nlp1_sparse_dummy_confun (e04ugm)`. (nag_opt_nlp1_sparse_dummy_confun (e04ugm) is included in the NAG Toolbox.) If there are nonlinear constraints, the first call to **confun** will occur before the first call to **objfun**.

```
[mode, f, fjac, user] = confun(mode, ncnln, njnln, nnzjac, x, fjac,
nstate, user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **confun**. Only the following values need be assigned:

mode $= 0$
f.

mode $= 1$
All available elements of **fjac**.

mode $= 2$
f and all available elements of **fjac**.

2: **ncnln** – INTEGER

n_N, the number of nonlinear constraints. These must be the first **ncnln** constraints in the problem.

3: **njnln** – INTEGER

n_1'', the number of nonlinear variables. These must be the first **njnln** variables in the problem.

4: **nnzjac** – INTEGER

The number of nonzero elements in the constraint Jacobian. Note that **nnzjac** will usually be less than **ncnln** $\times$ **njnln**.

5: **x**(**njnln**) – REAL (KIND=nag_wp) array

x, the vector of nonlinear Jacobian variables at which the nonlinear constraint functions and/or the available elements of the constraint Jacobian are to be evaluated.

6: **fjac**(**nnzjac**) – REAL (KIND=nag_wp) array

The elements of **fjac** are set to special values which enable nag_opt_nlp1_sparse_solve (e04ug) to detect whether they are changed by **confun**.

7: **nstate** – INTEGER

If **nstate** $= 1$, then nag_opt_nlp1_sparse_solve (e04ug) is calling **confun** for the first time. This argument setting allows you to save computation time if certain data must be read or calculated only once.

If **nstate** ≥ 2, then nag_opt_nlp1_sparse_solve (e04ug) is calling **confun** for the last time. This argument setting allows you to perform some additional computation on the final solution. In general, the last call to **confun** is made with **nstate** $= 2 +$ **ifail** (see Section 6).

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

8: **user** – INTEGER array

confun is called from nag_opt_nlp1_sparse_solve (e04ug) with the object supplied to nag_opt_nlp1_sparse_solve (e04ug).

Output Parameters

1: **mode** – INTEGER

You may set to a negative value as follows:

mode ≤ -2
The solution to the current problem is terminated and in this case nag_opt_nlp1_sparse_solve (e04ug) will terminate with **ifail** set to **mode**.

mode $= -1$
The nonlinear constraint functions cannot be calculated at the current x. nag_opt_nlp1_sparse_solve (e04ug) will then terminate with **ifail** $= -1$ unless this occurs during the linesearch; in this case, the linesearch will shorten the step and try again.

2: **f**(**ncnln**) – REAL (KIND=nag_wp) array

If **mode** $= 0$ or 2, **f**(i) must contain the value of the ith nonlinear constraint function at x.

3: **fjac**(**nnzjac**) – REAL (KIND=nag_wp) array

If **mode** $= 1$ or 2, **fjac** must return the available elements of the constraint Jacobian evaluated at x. These elements must be stored in exactly the same positions as implied by the definitions of the arrays **a**, **ha** and **ka**. If optional parameter **Derivative Level** $= 2$ or 3, the value of any constant Jacobian element not defined by **confun** will be obtained directly from **a**. Note that the function does not perform any internal checks for consistency (except indirectly via the optional parameter **Verify Level**), so great care is essential.

4: **user** – INTEGER array

2: **objfun** – SUBROUTINE, supplied by the NAG Library or the user.

objfun must calculate the nonlinear part of the objective function $f(x)$ and (optionally) its gradient $\left(= \dfrac{\partial f}{\partial x}\right)$ for a specified n_1' ($\leq n$) element vector x. If there are no nonlinear objective variables (i.e., **nonln** $= 0$), **objfun** will never be called by nag_opt_nlp1_sparse_solve (e04ug) and **objfun** may be the string `nag_opt_nlp1_sparse_dummy_objfun (e04ugn)`. (nag_opt_nlp1_sparse_dummy_objfun (e04ugn) is included in the NAG Toolbox.)

```
[mode, objf, objgrd, user] = objfun(mode, nonln, x, objgrd, nstate, user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$
objf.

mode $= 1$
All available elements of **objgrd**.

mode $= 2$
objf and all available elements of **objgrd**.

2: **nonln** – INTEGER

n_1', the number of nonlinear objective variables. These must be the first **nonln** variables in the problem.

3: **x**(**nonln**) – REAL (KIND=nag_wp) array

x, the vector of nonlinear variables at which the nonlinear part of the objective function and/or all available elements of its gradient are to be evaluated.

4: **objgrd**(**nonln**) – REAL (KIND=nag_wp) array

The elements of **objgrd** are set to special values which enable nag_opt_nlp1_sparse_solve (e04ug) to detect whether they are changed by **objfun**.

5: **nstate** – INTEGER

If **nstate** $= 1$, nag_opt_nlp1_sparse_solve (e04ug) is calling **objfun** for the first time. This argument setting allows you to save computation time if certain data must be read or calculated only once.

If **nstate** ≥ 2, nag_opt_nlp1_sparse_solve (e04ug) is calling **objfun** for the last time. This argument setting allows you to perform some additional computation on the final solution. In general, the last call to **objfun** is made with **nstate** $= 2 +$ **ifail** (see Section 6).

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

6: **user** – INTEGER array

objfun is called from nag_opt_nlp1_sparse_solve (e04ug) with the object supplied to nag_opt_nlp1_sparse_solve (e04ug).

Output Parameters

1: **mode** – INTEGER

You may set to a negative value as follows:

mode ≤ -2
The solution to the current problem is terminated and in this case nag_opt_nlp1_sparse_solve (e04ug) will terminate with **ifail** set to **mode**.

mode $= -1$
The nonlinear part of the objective function cannot be calculated at the current x. nag_opt_nlp1_sparse_solve (e04ug) will then terminate with **ifail** $= -1$ unless this occurs during the linesearch; in this case, the linesearch will shorten the step and try again.

2: **objf** – REAL (KIND=nag_wp)

If **mode** $= 0$ or 2, **objf** must be set to the value of the objective function at x.

3: **objgrd**(**nonln**) – REAL (KIND=nag_wp) array

If **mode** $= 1$ or 2, **objgrd** must return the available elements of the gradient evaluated at x.

4: **user** – INTEGER array

3: **n** – INTEGER

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

Constraint: **n** ≥ 1.

4: **m** – INTEGER

m, the number of general constraints (or slacks). This is the number of rows in A, including the free row (if any; see **iobj**). Note that A must contain at least one row. If your problem has no constraints, or only upper and lower bounds on the variables, then you must include a dummy 'free' row consisting of a single (zero) element subject to 'infinite' upper and lower bounds. Further details can be found under the descriptions for **iobj**, **nnz**, **a**, **ha**, **ka**, **bl** and **bu**.

Constraint: **m** ≥ 1.

5: **ncnln** – INTEGER

n_N, the number of nonlinear constraints.

Constraint: $0 \leq$ **ncnln** $\leq$ **m**.

6: **nonln** – INTEGER

n_1', the number of nonlinear objective variables. If the objective function is nonlinear, the leading n_1' columns of A belong to the nonlinear objective variables. (See also the description for **njnln**.)

Constraint: $0 \leq$ **nonln** $\leq$ **n**.

7: **njnln** – INTEGER

n_1'', the number of nonlinear Jacobian variables. If there are any nonlinear constraints, the leading n_1'' columns of A belong to the nonlinear Jacobian variables. If $n_1' > 0$ and $n_1'' > 0$, the nonlinear objective and Jacobian variables overlap. The total number of nonlinear variables is given by $\bar{n} = \max\left(n_1', n_1''\right)$.

Constraints:

if **ncnln** $= 0$, **njnln** $= 0$;
if **ncnln** > 0, $1 \leq$ **njnln** $\leq$ **n**.

8: **iobj** – INTEGER

If **iobj** $>$ **ncnln**, row **iobj** of A is a free row containing the nonzero elements of the linear part of the objective function.

iobj $= 0$
There is no free row.

iobj $= -1$
There is a dummy 'free' row.

Constraints:

if **iobj** > 0, **ncnln** $<$ **iobj** $\leq$ **m**;
otherwise **iobj** ≥ -1.

9: **a**(**nnz**) – REAL (KIND=nag_wp) array

The nonzero elements of the Jacobian matrix A, ordered by increasing column index. Since the constraint Jacobian matrix $J(x'')$ must always appear in the top left-hand corner of A, those elements in a column associated with any nonlinear constraints must come before any elements belonging to the linear constraint matrix G and the free row (if any; see **iobj**).

In general, **a** is partitioned into a nonlinear part and a linear part corresponding to the nonlinear variables and linear variables in the problem. Elements in the nonlinear part may be set to any

value (e.g., zero) because they are initialized at the first point that satisfies the linear constraints and the upper and lower bounds.

If **Derivative Level** $= 2$ or 3, the nonlinear part may also be used to store any constant Jacobian elements. Note that if **confun** does not define the constant Jacobian element **fjac**(i) then the missing value will be obtained directly from **a**(j) for some $j \geq i$.

If **Derivative Level** $= 0$ or 1, unassigned elements of **fjac** are *not* treated as constant; they are estimated by finite differences, at nontrivial expense.

The linear part must contain the nonzero elements of G and the free row (if any). If **iobj** $= -1$, set **a**$(1) = 0$. Elements with the same row and column indices are not allowed. (See also the descriptions for **ha** and **ka**.)

10: **ha**(**nnz**) – INTEGER array

ha(i) must contain the row index of the nonzero element stored in **a**(i), for $i = 1, 2, \ldots,$ **nnz**. The row indices for a column may be supplied in any order subject to the condition that those elements in a column associated with any nonlinear constraints must appear before those elements associated with any linear constraints (including the free row, if any). Note that **confun** must define the Jacobian elements in the same order. If **iobj** $= -1$, set **ha**$(1) = 1$.

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

11: **ka**(**n** + **1**) – INTEGER array

ka(j) must contain the index in **a** of the start of the jth column, for $j = 1, 2, \ldots,$ **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**(**n** + 1) = **nnz** + 1. If **iobj** $= -1$, set **ka**$(j) = 2$, for $j = 2, 3, \ldots,$ **n**.

Constraints:

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

12: **bl**(**n** + **m**) – REAL (KIND=nag_wp) 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, the next **ncnln** elements the bounds for the nonlinear constraints $F(x)$ (if any) and the next (**m** − **ncnln**) elements the bounds for the linear constraints Gx and the free row (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set **bl**$(j) \leq -bigbnd$. To specify the jth constraint as an *equality*, set **bl**$(j) =$ **bu**$(j) = \beta$, say, where $|\beta| < bigbnd$. If **iobj** $= -1$, set **bl**(**n** + abs(**iobj**)) $\leq$ $-bigbnd$.

Constraint: if **ncnln** $<$ **iobj** $\leq$ **m** or **iobj** $= -1$, **bl**(**n** + abs(**iobj**)) $\leq -bigbnd$

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

13: **bu**(**n** + **m**) – REAL (KIND=nag_wp) array

u, the upper bounds for all the variables and general constraints, in the following order. The first **n** elements of **bu** must contain the bounds on the variables x, the next **ncnln** elements the bounds for the nonlinear constraints $F(x)$ (if any) and the next (**m** − **ncnln**) elements the bounds for the linear constraints Gx and the free row (if any). To specify a nonexistent upper bound (i.e., $u_j = +\infty$), set **bu**$(j) \geq bigbnd$. To specify the jth constraint as an *equality*, set **bu**$(j) =$ **bl**$(j) = \beta$, say, where $|\beta| < bigbnd$. If **iobj** $= -1$, set **bu**(**n** + abs(**iobj**)) $\geq bigbnd$.

Constraints:

if **ncnln** $<$ **iobj** $\leq$ **m** or **iobj** $= -1$, **bu**(**n** + abs(**iobj**)) $\geq bigbnd$;
bl$(j) \leq$ **bu**(j), for $j = 1, 2, \ldots,$ **n** + **m**;
if **bl**$(j) =$ **bu**$(j) = \beta$, $|\beta| < bigbnd$.

14: **start** – CHARACTER(1)

Indicates how a starting basis is to be obtained.

start = 'C'
An internal Crash procedure will be used to choose an initial basis.

start = 'W'
A basis is already defined in **istate** and **ns** (probably from a previous call).

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

15: **names**(**nname**) – CHARACTER(8) array

Specifies the column and row names to be used in the printed output.

If **nname** = 1, **names** 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, the next **ncnln** elements must contain the names for the nonlinear rows (if any) and the next (**m** − **ncnln**) elements must contain the names for the linear rows (if any) to be used in the printed output. Note that the name for the free row or dummy 'free' row must be stored in **names**(**n** + abs(**iobj**)).

16: **ns** – INTEGER

n_S, the number of superbasics. It need not be specified if **start** = 'C', but must retain its value from a previous call when **start** = 'W'.

17: **xs**(**n** + **m**) – REAL (KIND=nag_wp) array

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

18: **istate**(**n** + **m**) – INTEGER 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) ≤ **xs**(j) ≤ **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 the function has been called previously with the same values of **n** and **m**, **istate** already contains satisfactory information.

Constraints:

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

19: **clamda**(**n** + **m**) – REAL (KIND=nag_wp) array

If **ncnln** > 0, **clamda**(j) must contain a Lagrange multiplier estimate for the jth nonlinear constraint $F_j(x)$, for $j =$ **n** $+ 1, \ldots,$ **n** + **ncnln**. If nothing special is known about the problem, or there is no wish to provide special information, you may set **clamda**$(j) = 0.0$. The remaining elements need not be set.

20: **lwsav**(**20**) – LOGICAL array
21: **iwsav**(**550**) – INTEGER array
22: **rwsav**(**550**) – REAL (KIND=nag_wp) array

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions nag_opt_nlp1_sparse_solve (e04ug), nag_opt_nlp1_sparse_option_string (e04uj).

5.2 Optional Input Parameters

1: **nz** – INTEGER

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 (including the Jacobian for any nonlinear constraints). If **iobj** $= -1$, set **nnz** $= 1$.

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

2: **nname** – INTEGER

Default: the dimension of the array **names**.

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

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: **leniz** – INTEGER

Default: $\max(500,$ **n** $+$ **m**$)$

The dimension of the array **iz**.

Constraint: **leniz** $\geq \max(500,$ **n** $+$ **m**$)$.

4: **lenz** – INTEGER

The dimension of the array **z**.

Constraint: **lenz** ≥ 500.

The amounts of workspace provided (i.e., **leniz** and **lenz**) and required (i.e., **miniz** and **minz**) are (by default) output on the current advisory message unit (as defined by nag_file_set_unit_advisory (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** set to $\max(500,$ **n** $+$ **m**$)$ and/or **lenz** set to 500. (nag_opt_nlp1_sparse_solve (e04ug) will then terminate with **ifail** $= 15$ or 16.)

5: **user** – INTEGER array

user is not used by nag_opt_nlp1_sparse_solve (e04ug), but is passed to **confun** and **objfun**. 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 Output Parameters

1: **a**(**nnz**) – REAL (KIND=nag_wp) array

Elements in the nonlinear part corresponding to nonlinear Jacobian variables are overwritten.

2: **ns** – INTEGER

The final number of superbasics.

3: **xs**(**n** + **m**) – REAL (KIND=nag_wp) array

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

4: **istate**(**n** + **m**) – INTEGER 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 **Minor Feasibility Tolerance**. Note that if scaling is specified, the optional parameter **Minor 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 **Minor 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 scaling was not used).

5: **clamda**(**n** + **m**) – REAL (KIND=nag_wp) array

A set of Lagrange multipliers for the bounds on the variables (*reduced costs*) and the general constraints (*shadow costs*). More precisely, the first **n** elements contain the multipliers for the bounds on the variables, the next **ncnln** elements contain the multipliers for the nonlinear constraints $F(x)$ (if any) and the next (**m** − **ncnln**) elements contain the multipliers for the linear constraints Gx and the free row (if any).

6: **miniz** – INTEGER

The minimum value of **leniz** required to start solving the problem. If **ifail** $= 12$, nag_opt_nlp1_sparse_solve (e04ug) 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.)

7: **minz** – INTEGER

The minimum value of **lenz** required to start solving the problem. If **ifail** $= 13$, nag_opt_nlp1_sparse_solve (e04ug) 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.)

8: **ninf** – INTEGER

The number of constraints that lie outside their bounds by more than the value of the optional parameter **Minor Feasibility Tolerance**.

If the *linear* constraints are infeasible, the sum of the infeasibilities of the linear constraints is minimized subject to the upper and lower bounds being satisfied. In this case, **ninf** contains the number of elements of Gx that lie outside their upper or lower bounds. Note that the nonlinear constraints are not evaluated.

Otherwise, the sum of the infeasibilities of the *nonlinear* constraints is minimized subject to the linear constraints and the upper and lower bounds being satisfied. In this case, **ninf** contains the number of elements of $F(x)$ that lie outside their upper or lower bounds.

9: **sinf** – REAL (KIND=nag_wp)

The sum of the infeasibilities of constraints that lie outside their bounds by more than the value of the optional parameter **Minor Feasibility Tolerance**.

10: **obj** – REAL (KIND=nag_wp)

The value of the objective function.

11: **user** – INTEGER array

12: **lwsav**(**20**) – LOGICAL array
13: **iwsav**(**550**) – INTEGER array
14: **rwsav**(**550**) – REAL (KIND=nag_wp) array

15: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

nag_opt_nlp1_sparse_solve (e04ug) returns with **ifail** $= 0$ if the iterates have converged to a point x that satisfies the first-order Kuhn–Karesh–Tucker conditions (see Section 9.1) to the accuracy requested by the optional parameters **Major Feasibility Tolerance** (default value $= \sqrt{\epsilon}$) and **Major Optimality Tolerance** (default value $= \sqrt{\epsilon}$).

6 Error Indicators and Warnings

Note: nag_opt_nlp1_sparse_solve (e04ug) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail < 0 (*warning*)

A negative value of **ifail** indicates an exit from nag_opt_nlp1_sparse_solve (e04ug) because you set **mode** < 0 in **objfun** or **confun**. The value of **ifail** will be the same as your setting of **mode**.

ifail $= 1$ (*warning*)

The problem is infeasible. The general constraints cannot all be satisfied simultaneously to within the values of the optional parameters **Major Feasibility Tolerance** (default value $= \sqrt{\epsilon}$) and **Minor Feasibility Tolerance** (default value $= \sqrt{\epsilon}$).

ifail $= 2$ (*warning*)

The problem is unbounded (or badly scaled). The objective function is not bounded below (or above in the case of maximization) in the feasible region because a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Add an upper or lower bound to the variable (whose index is printed by default by nag_opt_nlp1_sparse_solve (e04ug)) and rerun nag_opt_nlp1_sparse_solve (e04ug).

ifail $= 3$ (*warning*)

The problem may be unbounded. Check that the values of the optional parameters **Unbounded Objective** (default value $= 10^{15}$) and **Unbounded Step Size** (default value $= \max(bigbnd, 10^{20})$) are not too small. This exit also implies that the objective function is not bounded below (or above in the case of maximization) in the feasible region defined by expanding the bounds by the value of the optional parameter **Violation Limit** (default value $= 10.0$).

ifail $= 4$

Too many iterations. The values of the optional parameters **Major Iteration Limit** (default value $= 1000$) and/or **Iteration Limit** (default value $= 10000$) are too small.

ifail $= 5$ (*warning*)

Feasible solution found, but requested accuracy could not be achieved. Check that the value of the optional parameter **Major Optimality Tolerance** (default value $= \sqrt{\epsilon}$) is not too small (say, $< \epsilon$).

ifail $= 6$

The value of the optional parameter **Superbasics Limit** (default value $= \min(500, \bar{n} + 1)$) is too small.

ifail $= 7$

An input argument is invalid.

ifail $= 8$

The user-supplied derivatives of the objective function computed by **objfun** appear to be incorrect. Check that **objfun** has been coded correctly and that all relevant elements of the objective gradient have been assigned their correct values.

ifail $= 9$

The user-supplied derivatives of the nonlinear constraint functions computed by **confun** appear to be incorrect. Check that **confun** has been coded correctly and that all relevant elements of the nonlinear constraint Jacobian have been assigned their correct values.

ifail $= 10$ (*warning*)

The current point cannot be improved upon. Check that **objfun** and **confun** have been coded correctly and that they are consistent with the value of the optional parameter **Derivative Level** (default value $= 3$).

ifail $= 11$

Numerical error in trying to satisfy the linear constraints (or the linearized nonlinear constraints). The basis is very ill-conditioned.

ifail $= 12$

Not enough integer workspace for the basis factors. Increase **leniz** and rerun nag_opt_nlp1_sparse_solve (e04ug).

ifail $= 13$

Not enough real workspace for the basis factors. Increase **lenz** and rerun nag_opt_nlp1_sparse_solve (e04ug).

ifail $= 14$ (*warning*)

The basis is singular after 15 attempts to factorize it (and adding slacks where necessary). Either the problem is badly scaled or the value of the optional parameter **LU Factor Tolerance** (default value = 5.0 or 100.0) is too large.

ifail $= 15$ (*warning*)

Not enough integer workspace to start solving the problem. Increase **leniz** to at least **miniz** and rerun nag_opt_nlp1_sparse_solve (e04ug).

ifail $= 16$ (*warning*)

Not enough real workspace to start solving the problem. Increase **lenz** to at least **minz** and rerun nag_opt_nlp1_sparse_solve (e04ug).

ifail $= 17$

An unexpected error has occurred. Please contact NAG.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

If the value of the optional parameter **Major Optimality Tolerance** is set to 10^{-d} (default value $= \sqrt{\epsilon}$) and **ifail** $= 0$ on exit, then the final value of $f(x)$ should have approximately d correct significant digits.

8 Further Comments

This section contains a description of the printed output.

8.1 Major Iteration Printout

This section describes the intermediate printout and final printout produced by the major iterations of nag_opt_nlp1_sparse_solve (e04ug). The intermediate printout is a subset of the monitoring information produced by the function at every iteration (see Section 13). You can control the level of printed output (see the description of the optional parameter **Major Print Level**). Note that the intermediate printout and final printout are produced only if **Major Print Level** ≥ 10 (the default for nag_opt_nlp1_sparse_solve (e04ug), by default no output is produced by nag_opt_nlp1_sparse_solve (e04ug)).

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

`Maj` is the major iteration count.

`Mnr` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 11).

Note that `Mnr` may be greater than the optional parameter **Minor Iteration Limit** if some iterations are required for the feasibility phase.

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Merit Function` is the value of the augmented Lagrangian merit function (6) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 9.1). As the solution is approached, `Merit Function` will converge to the value of the objective function at the solution.

In elastic mode (see Section 11.2) then the merit function is a composite function involving the constraint violations weighted by the value of the optional parameter **Elastic Weight**.

If there are no nonlinear constraints present then this entry contains `Objective`, the value of the objective function $f(x)$. In this case, $f(x)$ will decrease monotonically to its optimal value.

`Feasibl` is the value of *rowerr*, the largest element of the scaled nonlinear constraint residual vector defined in the description of the optional parameter **Major Feasibility Tolerance**. The solution is regarded as 'feasible' if `Feasibl` is less than (or equal to) the optional parameter **Major Feasibility Tolerance**. `Feasibl` will be approximately zero in the neighbourhood of a solution.

If there are no nonlinear constraints present, all iterates are feasible and this entry is not printed.

`Optimal` is the value of *maxgap*, the largest element of the maximum complementarity gap vector defined in the description of the optional parameter **Major Optimality Tolerance**. The Lagrange multipliers are regarded as 'optimal' if `Optimal` is less than (or equal to) the optional parameter **Major Optimality Tolerance**. `Optimal` will be approximately zero in the neighbourhood of a solution.

`Cond Hz` is an estimate of the condition number of the reduced Hessian of the Lagrangian (not printed if **ncnln** and **nonln** are both zero). It is the square of the ratio between the largest and smallest diagonal elements of the upper triangular matrix R. This constitutes a lower bound on the condition number of the matrix $R^{\mathrm{T}}R$ that approximates the reduced Hessian. The larger this number, the more difficult the problem.

`PD` is a two-letter indication of the status of the convergence tests involving the feasibility and optimality of the iterates defined in the descriptions of the optional parameters **Major Feasibility Tolerance** and **Major Optimality Tolerance**. Each letter is `T` if the test is satisfied and `F` otherwise. The tests indicate whether the values of `Feasibl` and `Optimal` are sufficiently small. For example, `TF` or `TT` is printed if there are no nonlinear constraints present (since all iterates are feasible). If either indicator is `F` when nag_opt_nlp1_sparse_solve (e04ug) terminates with **ifail** $= 0$, you should check the solution carefully.

`M` is printed if an extra evaluation of user-supplied functions **objfun** and **confun** was needed in order to define an acceptable positive definite quasi-Newton update to the Hessian of the Lagrangian. This modification is only performed when there are nonlinear constraints present.

`m` is printed if, in addition, it was also necessary to modify the update to include an augmented Lagrangian term.

`s` is printed if a self-scaled BFGS (Broyden–Fletcher–Goldfarb–Shanno) update was performed. This update is always used when the Hessian approximation is diagonal and hence always follows a Hessian reset.

`S` is printed if, in addition, it was also necessary to modify the self-scaled update in order to maintain positive-definiteness.

`n` is printed if no positive definite BFGS update could be found, in which case the approximate Hessian is unchanged from the previous iteration.

`r` is printed if the approximate Hessian was reset after 10 consecutive major iterations in which no BFGS update could be made. The diagonal elements of the approximate Hessian are retained if at least one update has been performed since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`R` is printed if the approximate Hessian has been reset by discarding all but its diagonal elements. This reset will be forced periodically by the values of the optional parameters **Hessian Frequency** and **Hessian Updates**. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`l` is printed if the change in the norm of the variables was greater than the value defined by the optional parameter **Major Step Limit**. If this output occurs frequently during later iterations, it may be worthwhile increasing the value of **Major Step Limit**.

`c` is printed if central differences have been used to compute the unknown elements of the objective and constraint gradients. A switch to central differences is made if either the linesearch gives a small step, or x is close to being optimal. In some cases, it may be necessary to re-solve the QP subproblem with the central difference gradient and Jacobian.

`u` is printed if the QP subproblem was unbounded.

`t` is printed if the minor iterations were terminated after the number of iterations specified by the value of the optional parameter **Minor Iteration Limit** was reached.

`i` is printed if the QP subproblem was infeasible when the function was not in elastic mode. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem (8) (see Section 11.2). It is also printed if the minimizer of the elastic subproblem does not satisfy the linearized constraints when the function is already in elastic mode. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`w` is printed if a weak solution of the QP subproblem was found.

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** $= \mathbf{n} + \mathbf{m}$, the name supplied in **names**(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** $= 0$ 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 current value, there would be no change in the value of the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the Lagrange multipliers *might* also change.

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

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

`N` *Not precisely optimal*. The variable is nonbasic. Its reduced gradient is larger than the value of the optional parameter **Major Feasibility Tolerance**.

`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 non-negative if `State` is `LL`, non-positive 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 general constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', n replaced by m, $\mathbf{names}(j)$ replaced by $\mathbf{names}(n+j)$, $\mathbf{bl}(j)$ and $\mathbf{bu}(j)$ are replaced by $\mathbf{bl}(n+j)$ and $\mathbf{bu}(n+j)$ respectively. The heading is changed as follows:

`Constrnt` gives the name of the general constraint.

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

8.2 Minor Iteration Printout

This section describes the printout produced by the minor iterations of nag_opt_nlp1_sparse_solve (e04ug), which involve solving a QP subproblem at every major iteration. (Further details can be found in Section 9.1.) The printout is a subset of the monitoring information produced by the function at every iteration (see Section 13). You can control the level of printed output (see the description of the optional parameter **Minor Print Level**). Note that the printout is produced only if **Minor Print Level** ≥ 1 (default value $= 0$, which produces no output).

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

`Itn` is the iteration count.

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

`Ninf` is the number of infeasibilities. This will not increase unless the iterations are in elastic mode. `Ninf` will be zero during the optimality phase.

`Sinf` is the value of the sum of infeasibilities if `Ninf` is nonzero. This will be zero during the optimality phase.

`Objective` is the value of the current QP objective function when `Ninf` is zero and the iterations are not in elastic mode. The switch to elastic mode is indicated by a change in the heading to `Composite Obj`.

`Composite Obj` is the value of the composite objective function (9) when the iterations are in elastic mode. This function will decrease monotonically at each iteration.

`Norm rg` is the Euclidean norm of the reduced gradient of the QP objective function. During the optimality phase, this norm will be approximately zero after a unit step.

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

9 Example

This is a reformulation of Problem 74 in Hock and Schittkowski (1981) and involves the minimization of the nonlinear function

$$f(x) = 10^{-6}x_3^3 + \tfrac{2}{3} \times 10^{-6}x_4^3 + 3x_3 + 2x_4$$

subject to the bounds

$$\begin{aligned} -0.55 \leq x_1 &\leq 0.55, \\ -0.55 \leq x_2 &\leq 0.55, \\ 0 \leq x_3 &\leq 1200, \\ 0 \leq x_4 &\leq 1200, \end{aligned}$$

to the nonlinear constraints

$$\begin{aligned} 1000\sin(-x_1 - 0.25) &+ 1000\sin(-x_2 - 0.25) - x_3 &&= -894.8, \\ 1000\sin(x_1 - 0.25) &+ 1000\sin(x_1 - x_2 - 0.25) - x_4 &&= -894.8, \\ 1000\sin(x_2 - 0.25) &+ 1000\sin(x_2 - x_1 - 0.25) &&= -1294.8, \end{aligned}$$

and to the linear constraints

$$\begin{aligned} -x_1 + x_2 &\geq -0.55, \\ x_1 - x_2 &\geq -0.55. \end{aligned}$$

The initial point, which is infeasible, is

$$x_0 = \begin{pmatrix} 0, & 0, & 0, & 0 \end{pmatrix}^{\mathrm{T}},$$

and $f(x_0) = 0$.

The optimal solution (to five figures) is

$$x^* = (0.11887, -0.39623, 679.94, 1026.0)^{\mathrm{T}},$$

and $f(x^*) = 5126.4$. All the nonlinear constraints are active at the solution.

9.1 Program Text

```
function e04ug_example

fprintf('e04ug example results\n\n');

n     = nag_int(4);
m     = nag_int(6);
ncnln = nag_int(3);
nonln = nag_int(4);
njnln = nag_int(2);
iobj  = nag_int(6);
a  = [1e25; 1e25; 1e25;   1;   -1;
      1e25; 1e25; 1e25;  -1;    1;
         3;   -1;
        -1;    2];
ha = nag_int([ 1; 2; 3; 5; 4; 1; 2; 3; 5; 4; 6; 1; 2; 6]);
ka = nag_int([ 1;             6;            11;   13;    15]);
bl = [-0.55; -0.55;    0;    0;
                                -894.8; -894.8; -1294.8; -0.55; -0.55; -1e25];
bu = [ 0.55;  0.55; 1200; 1200;
                                -894.8; -894.8; -1294.8;  1e25;  1e25;  1e25];
start = 'C';
names = {'Varble 1'; 'Varble 2'; 'Varble 3'; 'Varble 4'; 'NlnCon 1'; ...
         'NlnCon 2'; 'NlnCon 3'; 'LinCon 1'; 'LinCon 2'; 'Free Row'};
ns = nag_int(0);
xs     = [ 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
istate(1:10) = nag_int(0);
clamda = [ 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
leniz  = nag_int(1000);
lenz   = nag_int(1000);
```

```
% Catch warnings and assume ifail=3,5 gives a good estimate
wstat = warning();
warning('OFF');

[cwsav,lwsav,iwsav,rwsav,ifail] = e04wb('e04ug');
disp('First call to e04ug')
[a, ns, xs, istate, clamda, miniz, minz, ninf, sinf, obj, user, lwsav, ...
   iwsav, rwsav, ifail] = ...
     e04ug(...
           @confun, @objfun, n, m, ncnln, nonln, njnln, ...
            iobj, a, ha, ka, bl, bu, start, names, ns, ...
            xs, istate, clamda, lwsav, iwsav, rwsav);
if (ifail == 15 || ifail == 16)
  % Default amount of workspace is insufficient, use values bigger than those
  % returned in minz and miniz
  minz  = 10*minz;
  miniz = 10*miniz;
  fprintf('  returned with ifail = %d,\n',ifail);
  fprintf('  increasing minz to %d and miniz to %d and calling again\n', ...
          minz, miniz);
  fprintf('\nSecond call to e04ug\n')
  [a, ns, xs, istate, clamda, miniz, minz, ninf, sinf, obj, user, lwsav, ...
     iwsav, rwsav, ifail] = ...
      e04ug(...
            @confun, @objfun, n, m, ncnln, nonln, njnln, ...
            iobj, a, ha, ka, bl, bu, start, names, ns, ...
            xs, istate, clamda, lwsav, iwsav, rwsav, ...
            'lenz', minz, 'leniz', miniz);
end
if (ifail==0)
  fprintf('\nMinimum found at x: ');
  fprintf(' %9.4f',xs(1:n));
  fprintf('\nMinimum value     :  %9.4f\n\n',obj);
else
  fprintf('\n Error: e04ug returns ifail = %d\n',ifail);
end

warning(wstat);

function [mode, f, fjac, user] = ...
   confun(mode, ncnln, njnln, nnzjac, x, fjac, nstate, user)
  f = zeros(ncnln, 1);

  x1 = -x(1) - 0.25;
  x2 = -x(2) - 0.25;
  x3 =  x(1) - 0.25;
  x4 =  x(2) - 0.25;
  x5 =  x(1) - x(2) - 0.25;
  x6 =  x(2) - x(1) - 0.25;
  if (mode == 0 || mode == 2)
    f(1) = 1000*sin(x1) + 1000*sin(x2);
    f(2) = 1000*sin(x3) + 1000*sin(x5);
    f(3) = 1000*sin(x6) + 1000*sin(x4);
  end

  if (mode == 1 || mode == 2)
%   nonlinear jacobian elements for column 1.
    fjac(1) = -1000*cos(x1);
    fjac(2) =  1000*cos(x3) + 1000*cos(x5);
    fjac(3) = -1000*cos(x6);
%   nonlinear jacobian elements for column 2.
    fjac(4) = -1000*cos(x2);
    fjac(5) = -1000*cos(x5);
    fjac(6) =  1000*cos(x6) + 1000*cos(x4);
  end

function [mode, objf, objgrd, user] = ...
    objfun(mode, nonln, x, objgrd, nstate, user)

  if (mode == 0 || mode == 2)
    objf = 1.0e-6*(x(3)^3 + 2*x(4)^3/3);
```

```
  end

  if (mode == 1 || mode == 2)
    objgrd(1) = 0;
    objgrd(2) = 0;
    objgrd(3) = 3.0e-6*x(3)^2;
    objgrd(4) = 2.0e-6*x(4)^2;
  end
```

9.2 Program Results

```
     e04ug example results

First call to e04ug
  returned with ifail = 15,
  increasing minz to 7580 and miniz to 6280 and calling again

Second call to e04ug

Minimum found at x:     0.1189   -0.3962  679.9453 1026.0671
Minimum value     :  5126.4981
```

Note: *the remainder of this document is intended for more advanced users. Section 11 contains a detailed description of the algorithm which may be needed in order to understand Section 12 and Section 13. Section 12 describes the optional parameters which may be set by calls to nag_opt_nlp1_sparse_option_string (e04uj). Section 13 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 nag_opt_nlp1_sparse_solve (e04ug).

10.1 Overview

Here we briefly summarise the main features of the method and introduce some terminology. Where possible, explicit reference is made to the names of variables that are arguments of the function or appear in the printed output. Further details can be found in Gill *et al.* (2002).

At a solution of (1), some of the constraints will be *active*, i.e., satisfied exactly. Let

$$r(x) = \begin{pmatrix} x \\ F(x) \\ Gx \end{pmatrix}$$

and $\mathcal{G}$ denote the set of indices of $r(x)$ corresponding to active constraints at an arbitrary point x. Let $r'_j(x)$ denote the usual *derivative* of $r_j(x)$, which is the row vector of first partial derivatives of $r_j(x)$ (see Ortega and Rheinboldt (1970)). The vector $r'_j(x)$ comprises the jth row of $r'(x)$ so that

$$r'(x) = \begin{pmatrix} I \\ J(x) \\ G \end{pmatrix},$$

where $J(x)$ is the Jacobian of $F(x)$.

A point x is a *first-order Kuhn–Karesh–Tucker (KKT) point* for (1) (see Powell (1974)) if the following conditions hold:

(a) x is feasible;

(b) there exists a vector λ (*the Lagrange multiplier vector for the bound and general constraints*) such that

$$g(x) = r'(x)^{\mathrm{T}}\lambda = \begin{pmatrix} I & J(x)^{\mathrm{T}} & G^{\mathrm{T}} \end{pmatrix}\lambda, \tag{4}$$

where g is the gradient of f evaluated at x;

(c) the Lagrange multiplier λ_j associated with the jth constraint satisfies $\lambda_j = 0$ if $l_j < r_j(x) < u_j$; $\lambda_j \geq 0$ if $l_j = r_j(x)$; $\lambda_j \leq 0$ if $r_j(x) = u_j$; and λ_j can have any value if $l_j = u_j$.

An equivalent statement of the condition (4) is

$$Z^{\mathrm{T}}g(x) = 0,$$

where Z is a matrix defined as follows. Consider the set N of vectors orthogonal to the gradients of the active constraints, i.e.,

$$N = \left\{ z \quad | \quad r_j'(x)z = 0 \quad \text{for all} \quad j \in \mathcal{G} \right\}.$$

The columns of Z may then be taken as any basis for the vector space N. The vector $Z^{\mathrm{T}}g$ is termed the *reduced gradient* of f at x. Certain additional conditions must be satisfied in order for a first-order KKT point to be a solution of (1) (see Powell (1974)).

The basic structure of nag_opt_nlp1_sparse_solve (e04ug) involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that satisfy the linear constraints and converge to a point x^* that satisfies the first-order KKT optimality conditions. At each iterate a QP subproblem is used to generate a search direction towards the next iterate (x_{k+1}). The constraints of the subproblem are formed from the linear constraints $Gx - s_L = 0$ and the nonlinear constraint linearization

$$F(x_k) + F'(x_k)(x - x_k) - s_N = 0,$$

where $F'(x_k)$ denotes the *Jacobian matrix*, whose rows are the first partial derivatives of $F(x)$ evaluated at the point x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{aligned} F'(x_k)x - s_N &= -F(x_k) + F'(x_k)x_k, \\ Gx - s_L &= 0, \end{aligned}$$

where x and $s = (s_N, s_L)^{\mathrm{T}}$ are bounded above and below by u and l as before. If the m by n matrix A and m element vector b are defined as

$$A = \begin{pmatrix} F'(x_k) \\ G \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -F(x_k) + F'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} q(x) \quad \text{subject to} \quad Ax - s = b, \quad l \leq \begin{Bmatrix} x \\ s \end{Bmatrix} \leq u, \tag{5}$$

where $q(x)$ is a quadratic approximation to a modified Lagrangian function (see Gill *et al.* (2002)).

The linear constraint matrix A is stored in the arrays **a**, **ha** and **ka** (see Section 5). This allows you to specify the sparsity pattern of nonzero elements in $F'(x)$ and G and to identify any nonzero elements that remain constant throughout the minimization.

Solving the QP subproblem is itself an iterative procedure, with the *minor* iterations of an SQP method being the iterations of the QP method. At each minor iteration, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

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

where the *basis matrix* B is square and nonsingular. The elements of x_B, x_S and x_N are called the *basic*, *superbasic* and *nonbasic* variables respectively; 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 bounds, while the nonbasic variables will be equal to one of their upper or lower bounds. At each minor 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 QP objective function $q(x)$ or sum of infeasibilities (as appropriate). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. 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 if there are no nonlinear constraints in (1) and $f(x)$ is linear.

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 decreased by one.

Associated with each of the m equality constraints $Ax - s = b$ 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 $q(x)$; the reduced gradients for the slack variables s are the dual variables π. The QP subproblem (5) 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 other variables (including superbasics). In practice, an *approximate* QP solution is found by slightly relaxing these conditions on d_j (see the description of the optional parameter **Minor Optimality Tolerance**).

After a QP subproblem has been solved, new estimates of the solution to (1) are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^{\mathrm{T}}(F(x) - s_N) + \tfrac{1}{2}(F(x) - s_N)^{\mathrm{T}} D (F(x) - s_N), \tag{6}$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current estimate of the solution and $(\hat{x}, \hat{s}, \hat{\pi})$ denotes the optimal QP solution, the linesearch determines a step α_k (where $0 < \alpha_k \leq 1$) such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \hat{x}_k - x_k \\ \hat{s}_k - s_k \\ \hat{\pi}_k - \pi_k \end{pmatrix}$$

produces a *sufficient decrease* in the merit function (6). When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ (see Gill *et al.* (1992)). As in nag_opt_nlp2_solve (e04wd), s_N is adjusted to minimize the merit function as a function of s before the solution of the QP subproblem. Further details can be found in Eldersveld (1991) and Gill *et al.* (1986).

10.2 Treatment of Constraint Infeasibilities

nag_opt_nlp1_sparse_solve (e04ug) makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

$$\underset{x,v,w}{\text{minimize}} e^{\mathrm{T}}(v + w) \quad \text{subject to} \quad l \leq \begin{Bmatrix} x \\ Gx - v + w \end{Bmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \tag{7}$$

where $e = (1, 1, \ldots, 1)^{\mathrm{T}}$. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called *elastic programming*.)

If the linear constraints are infeasible (i.e., $v \neq 0$ or $w \neq 0$), the function terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates will satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which $f(x)$ and $F(x)$ can be safely evaluated.) The function then proceeds to solve (1) as given, using search directions obtained from a sequence of QP subproblems (5). Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to linearized constraints. An augmented Lagrangian merit function (6) is reduced along each search direction to ensure convergence from any starting point.

The function enters ‘elastic’ mode if the QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become ‘large’) by solving a problem of the form

$$\underset{x,v,w}{\text{minimize}} \bar{f}(x,v,w) \quad \text{subject to} \quad l \leq \begin{Bmatrix} x \\ F(x) - v + w \\ Gx \end{Bmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \tag{8}$$

where

$$\bar{f}(x,v,w) = f(x) + \gamma e^{\mathrm{T}}(v+w) \tag{9}$$

is called a *composite objective* and γ is a non-negative argument (the *elastic weight*). If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar l_1 formulation of (1) is fundamental to the $\mathrm{Sl}_1\mathrm{QP}$ algorithm of Fletcher (1984). See also Conn (1973).

11 Optional Parameters

Several optional parameters in nag_opt_nlp1_sparse_solve (e04ug) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_opt_nlp1_sparse_solve (e04ug) 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.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 12.1.

Central Difference Interval
Check Frequency
Crash Option
Crash Tolerance
Defaults
Derivative Level
Derivative Linesearch
Elastic Weight
Expand Frequency
Factorization Frequency
Feasibility Tolerance
Feasible Exit
Feasible Point
Forward Difference Interval
Function Precision
Hessian Frequency
Hessian Full Memory
Hessian Limited Memory
Hessian Updates
Infeasible Exit
Infinite Bound Size
Iteration Limit
Linesearch Tolerance
List
LU Density Tolerance
LU Factor Tolerance

LU Singularity Tolerance
LU Update Tolerance
Major Feasibility Tolerance
Major Iteration Limit
Major Optimality Tolerance
Major Print Level
Major Step Limit
Maximize
Minimize
Minor Feasibility Tolerance
Minor Iteration Limit
Minor Optimality Tolerance
Minor Print Level
Monitoring File
Nolist
Nonderivative Linesearch
Optimality Tolerance
Partial Price
Pivot Tolerance
Print Level
Scale Option
Scale Tolerance
Start Constraint Check At Column
Start Objective Check At Column
Stop Constraint Check At Column
Stop Objective Check At Column
Superbasics Limit
Unbounded Objective
Unbounded Step Size
Verify Level
Violation Limit

Optional parameters may be specified by calling nag_opt_nlp1_sparse_option_string (e04uj) before a call to nag_opt_nlp1_sparse_solve (e04ug).

nag_opt_nlp1_sparse_option_string (e04uj) can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
[lwsav, iwsav, rwsav, inform] = e04uj('Print Level = 5', lwsav, iwsav,
rwsav);
```

nag_opt_nlp1_sparse_option_string (e04uj) 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 nag_opt_nlp1_sparse_solve (e04ug) (unless they define invalid values) and so remain in effect for subsequent calls to nag_opt_nlp1_sparse_solve (e04ug) from the calling program (unless altered by you).

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined (if no characters of an optional qualifier are underlined, the qualifier may be omitted);

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value is used whenever the condition $|i| \geq 100000000$ is satisfied and where the symbol ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (x02aj)).

Keywords and character values are case and white space insensitive.

Central Difference Interval r Default $= \sqrt[3]{\textbf{Function Precision}}$

Note that this option does not apply when **Derivative Level** $= 3$.

The value of r is used near an optimal solution in order to obtain more accurate (but more expensive) estimates of gradients. This requires twice as many function evaluations as compared to using forward differences (see optional parameter **Forward Difference Interval**). The interval used for the jth variable is $h_j = r\left(1 + \left|x_j\right|\right)$. The resulting gradient estimates should be accurate to $O\left(r^2\right)$, unless the functions are badly scaled. The switch to central differences is indicated by c at the end of each line of intermediate printout produced by the major iterations (see Section 9.1). See Gill *et al.* (1981) for a discussion of the accuracy in finite difference approximations.

If $r \leq 0$, the default value is used.

Check Frequency i Default $= 60$

Every ith minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution (x, s) satisfies the general linear constraints (including any linearized nonlinear constraints). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. If the largest element of the residual vector $r = b - Ax + s$ is judged to be too large, the current basis is refactorized and the basic variables recomputed to satisfy the general 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 $= 0$ or 3

The default value of i is 0 if there are any nonlinear constraints and 3 otherwise. 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 of A are initially eligible for the basis and how many times the Crash procedure is called. Columns of $-I$ are used to pad the basis where necessary. The possible choices for i are the following.

i	**Meaning**
0	The initial basis contains only slack variables: $B = I$.
1	The Crash procedure is called once (looking for a triangular basis in all rows and columns of A).
2	The Crash procedure is called twice (if there are any nonlinear constraints). The first call looks for a triangular basis in linear rows and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and the Crash procedure is called again to find a triangular basis in the nonlinear rows (whilst retaining the current basis for linear rows).
3	The Crash procedure is called up to three times (if there are any nonlinear constraints). The first two calls treat linear *equality* constraints and linear *inequality* constraints separately. The Jacobian is then evaluated for the first major iteration and the Crash procedure is called again to find a triangular basis in the nonlinear rows (whilst retaining the current basis for linear rows).

If $i < 0$ or $i > 3$, the default value is used.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, 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 $= 0.1$

The value r ($0 \leq r < 1$) allows the Crash procedure to ignore certain 'small' nonzero elements in the columns of A while searching for a triangular basis. If $a_{\max}$ is the largest element in the jth column, other nonzeros a_{ij} in the column are ignored if $\left|a_{ij}\right| \leq a_{\max} \times r$.

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 containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached earlier on 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.

Derivative Level i Default $= 3$

This parameter indicates which nonlinear function gradients are provided in user-supplied functions **objfun** and **confun**. The possible choices for i are the following.

i	**Meaning**
3	All elements of the objective gradient and the constraint Jacobian are provided.
2	All elements of the constraint Jacobian are provided, but some (or all) elements of the objective gradient are not specified.
1	All elements of the objective gradient are provided, but some (or all) elements of the constraint Jacobian are not specified.
0	Some (or all) elements of both the objective gradient and the constraint Jacobian are not specified.

The default value $i = 3$ should be used whenever possible. It is the most reliable and will usually be the most efficient.

If $i = 0$ or 2, nag_opt_nlp1_sparse_solve (e04ug) will *estimate* the unspecified elements of the objective gradient, using finite differences. This may simplify the coding of **objfun**. However, the computation of finite difference approximations usually increases the total run-time substantially (since a call to **objfun** is required for each unspecified element) and there is less assurance that an acceptable solution will be found.

If $i = 0$ or 1, nag_opt_nlp1_sparse_solve (e04ug) will approximate unspecified elements of the constraint Jacobian. For each column of the Jacobian, one call to **confun** is needed to estimate all unspecified elements in that column (if any). For example, if the sparsity pattern of the Jacobian has the form

$$\begin{pmatrix} * & * & & * \\ & ? & ? & \\ * & & ? & \\ & * & & * \end{pmatrix}$$

where '$*$' indicates an element provided and '?' indicates an unspecified element, nag_opt_nlp1_sparse_solve (e04ug) will call **confun** twice: once to estimate the missing element in column 2 and again to estimate the two missing elements in column 3. (Since columns 1 and 4 are known, they require no calls to **confun**.)

At times, central differences are used rather than forward differences, in which case twice as many calls to **objfun** and **confun** are needed. (The switch to central differences is not under your control.)

If $i < 0$ or $i > 3$, the default value is used.

Derivative Linesearch Default
Nonderivative Linesearch

At each major iteration, a linesearch is used to improve the value of the Lagrangian merit function (6). The default linesearch uses safeguarded cubic interpolation and requires both function and gradient values in order to compute estimates of the step α_k. If some analytic derivatives are not provided or optional parameter **Nonderivative Linesearch** is specified, a linesearch based upon safeguarded quadratic interpolation (which does not require the evaluation or approximation of any gradients) is used instead.

A nonderivative linesearch can be slightly less robust on difficult problems and it is recommended that the default be used if the functions and their derivatives can be computed at approximately the same cost. If the gradients are very expensive to compute relative to the functions however, a nonderivative linesearch may result in a significant decrease in the total run-time.

If optional parameter **Nonderivative Linesearch** is selected, nag_opt_nlp1_sparse_solve (e04ug) signals the evaluation of the linesearch by calling **objfun** and **confun** with **mode** $= 0$. Once the linesearch is complete, the nonlinear functions are re-evaluated with **mode** $= 2$. If the potential savings offered by a nonderivative linesearch are to be fully realised, it is essential that **objfun** and **confun** be coded so that no derivatives are computed when **mode** $= 0$.

Elastic Weight r Default $= 1.0$ or 100.0

The default value of r is 100.0 if there are any nonlinear constraints and 1.0 otherwise.

This option defines the initial weight γ associated with problem (8).

At any given major iteration k, elastic mode is entered if the QP subproblem is infeasible or the QP dual variables (Lagrange multipliers) are larger in magnitude than $r \times \left(1 + \|g(x_k)\|_2\right)$, where g is the objective gradient. In either case, the QP subproblem is resolved in elastic mode with $\gamma = r \times \left(1 + \|g(x_k)\|_2\right)$.

Thereafter, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem (8), but not feasible for problem (1). After the pth increase, $\gamma = r \times 10^p \times \left(1 + \|g(x_{k_1})\|_2\right)$, where x_{k_1} is the iterate at which γ was first needed.

If $r < 0$, the default value is used.

Expand Frequency i Default $= 10000$

This option is part of the EXPAND anti-cycling procedure due to Gill *et al.* (1989), which is designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the constraints by a small amount. Suppose that the value of optional parameter **Minor Feasibility Tolerance** is δ. Over a period of i iterations, the feasibility tolerance actually used by nag_opt_nlp1_sparse_solve (e04ug) (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For nonlinear models, 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 $= 50$ or 100

The default value of i is 50 if there are any nonlinear constraints and 100 otherwise.

If $i > 0$, at most i basis changes will occur between factorizations of the basis matrix.

For linear problems, the basis factors are usually updated at every iteration. The default value $i = 100$ is reasonable for typical problems, particularly those that are extremely sparse and well-scaled.

When the objective function is nonlinear, 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 general constraints 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.

Infeasible Exit Default
Feasible Exit

Note that this option is ignored if the value of optional parameter **Major Iteration Limit** is exceeded, or the linear constraints are infeasible.

If termination is about to occur at a point that does not satisfy the nonlinear constraints and optional parameter **Feasible Exit** is selected, this option requests that additional iterations be performed in order to find a feasible point (if any) for the nonlinear constraints. This involves solving a feasible point problem in which the objective function is omitted.

Otherwise, this option requests no additional iterations be performed.

Minimize Default
Maximize
Feasible Point

If optional parameter **Feasible Point** is selected, this option attempts to find a feasible point (if any) for the nonlinear constraints by omitting the objective function. It can also be used to check whether the nonlinear constraints are feasible.

Otherwise, 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 will be reversed.

Forward Difference Interval r Default $= \sqrt{\textbf{Function Precision}}$

This option defines an interval used to estimate derivatives by forward differences in the following circumstances:

(a) For verifying the objective and/or constraint gradients (see the description of the optional parameter **Verify Level**).

(b) For estimating unspecified elements of the objective gradient and/or the constraint Jacobian.

A derivative with respect to x_j is estimated by perturbing that element of x to the value $x_j + r\left(1 + \left|x_j\right|\right)$ and then evaluating $f(x)$ and/or $F(x)$ (as appropriate) at the perturbed point. The resulting gradient estimates should be accurate to $O(r)$, unless the functions are badly scaled. Judicious alteration of r may sometimes lead to greater accuracy. See Gill *et al.* (1981) for a discussion of the accuracy in finite difference approximations.

If $r \leq 0$, the default value is used.

Function Precision r Default $= \epsilon^{0.8}$

This parameter defines the *relative function precision* ϵ_r, which is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ (or $F_i(x)$)

is computed as 1000.56789 for some relevant x and the first 6 significant digits are known to be correct then the appropriate value for ϵ_r would be 10^{-6}.

Ideally the functions $f(x)$ or $F_i(x)$ should have magnitude of order 1. If all functions are substantially *less* than 1 in magnitude, ϵ_r should be the *absolute* precision. For example, if $f(x)$ (or $F_i(x)$) is computed as $1.23456789 \times 10^{-4}$ for some relevant x and the first 6 significant digits are known to be correct then the appropriate value for ϵ_r would be 10^{-10}.

The choice of ϵ_r can be quite complicated for badly scaled problems; see Chapter 8 of Gill *et al.* (1981) for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy.

In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide few digits of precision at reasonable cost. Specifying an appropriate value of r may therefore lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

If $r < \epsilon$ or $r \geq 1$, the default value is used.

Hessian Frequency i Default $= 99999999$

This option forces the approximate Hessian formed from i BFGS updates to be reset to the identity matrix upon completion of a major iteration. It is intended to be used in conjunction with optional parameter **Hessian Full Memory**.

If $i \leq 0$, the default value is used and effectively no resets occur.

Hessian Full Memory Default when $\bar{n} < 75$
Hessian Limited Memory Default when $\bar{n} \geq 75$

These options specify the method for storing and updating the quasi-Newton approximation to the Hessian of the Lagrangian function.

If **Hessian Full Memory** is specified, the approximate Hessian is treated as a dense matrix and BFGS quasi-Newton updates are applied explicitly. This is most efficient when the total number of nonlinear variables is not too large (say, $\bar{n} < 75$). In this case, the storage requirement is fixed and you can expect 1-step Q-superlinear convergence to the solution.

Hessian Limited Memory should only be specified when $\bar{n}$ is very large. In this case a limited memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

Note that if **Hessian Frequency** $= 20$ is used in conjunction with **Hessian Full Memory**, the effect will be similar to using **Hessian Limited Memory** in conjunction with **Hessian Updates** $= 20$, except that the latter will retain the current diagonal during resets.

Hessian Updates i Default $= 20$ or 99999999

The default value of i is 20 when **Hessian Limited Memory** is in effect and 99999999 when **Hessian Full Memory** is in effect, in which case no updates are performed.

If **Hessian Limited Memory** is in effect, this option defines the maximum number of pairs of Hessian update vectors that are to be used to define the quasi-Newton approximate Hessian. Once the limit of i updates is reached, all but the diagonal elements of the accumulated updates are discarded and the process starts again. Broadly speaking, the more updates that are stored, the better the quality of the approximate Hessian. On the other hand, the more vectors that are stored, the greater the cost of each QP iteration.

The default value of i is likely to give a robust algorithm without significant expense, but faster convergence may be obtained with far fewer updates (e.g., $i = 5$).

If $i < 0$, the default value is used.

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 $+\infty$ (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as $-\infty$).

If $r \leq 0$, the default value is used.

Iteration Limit i Default $= 10000$

The value of i specifies the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. (See also the description of the optional parameters **Major Iteration Limit** and **Minor Iteration Limit**.)

If $i < 0$, the default value is used.

Linesearch Tolerance r Default $= 0.9$

This option controls the accuracy with which a step length will be located along the direction of search at each iteration. At the start of each linesearch a target directional derivative for the Lagrangian merit function is identified. The value of r therefore determines the accuracy to which this target value is approximated.

The default value $r = 0.9$ requests an inaccurate search and is appropriate for most problems, particularly those with any nonlinear constraints.

If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try $r = 0.1, 0.01$ or 0.001. The number of major iterations required to solve the problem might decrease.

If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. If **Derivative Level** $= 3$, try $r = 0.99$. (The number of major iterations required to solve the problem might increase, but the total number of function evaluations may decrease enough to compensate.)

If **Derivative Level** < 3, a moderately accurate search may be appropriate; try $r = 0.5$. Each search will (typically) require only $1 - 5$ function values, but many function calls will then be needed to estimate the missing gradients for the next iteration.

If $r < 0$ or $r \geq 1$, the default value is used.

List
Nolist Default for nag_opt_nlp1_sparse_solve (e04ug)
$=$ **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 Density Tolerance r_1 Default $= 0.6$
LU Singularity Tolerance r_2 Default $= \epsilon^{0.67}$

If $r_1 > 0$, r_1 defines the density tolerance used during the 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 r_1, the Markowitz strategy for choosing pivots is terminated. The remaining matrix is then factorized using a dense LU procedure. Increasing the value of r_1 towards unity may give slightly sparser LU factors, with a slight increase in factorization time. If $r_1 \leq 0$, the default value is used.

If $r_2 > 0$, r_2 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 $\left|u_{jj}\right| \leq r_2$ or $\left|u_{jj}\right| < r_2 \times \max_i \left|u_{ij}\right|$, the jth column of the basis is replaced by the corresponding slack variable. This is most likely to occur when **start** $=$ 'W' (see Section 5), or at the start of a major iteration. If $r_2 \leq 0$, the default value is used.

In some cases, the Jacobian matrix may converge to values that make the basis exactly singular (e.g., a whole row of the Jacobian matrix 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 $r_2 = 0.00001$ (say) may therefore help cause a judicious change of basis in such situations.

LU Factor Tolerance r_1 Default $= 5.0$ or 100.0
LU Update Tolerance r_2 Default $= 5.0$ or 10.0

The default value of r_1 is 5.0 if there are any nonlinear constraints and 100.0 otherwise. The default value of r_2 is 5.0 if there are any nonlinear constraints and 10.0 otherwise.

If $r_1 \geq 1$ and $r_2 \geq 1$, the values of r_1 and r_2 affect the stability and sparsity of the basis factorization $B = LU$, during refactorization and updating, 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 satisfy $|\mu| \leq r_i$. Smaller values of r_i favour stability, while larger values favour sparsity. The default values of r_1 and r_2 usually strike a good compromise. For large and relatively dense problems, setting $r_1 = 10.0$ or 5.0 (say) may give a marked improvement in sparsity without impairing stability to a serious degree. Note that for problems involving band matrices, it may be necessary to reduce r_1 and/or r_2 in order to achieve stability.

If $r_1 < 1$ or $r_2 < 1$, the appropriate default value is used.

Major Feasibility Tolerance r Default $= \sqrt{\epsilon}$

This option specifies how accurately the nonlinear constraints should be satisfied. The default value is appropriate when the linear and nonlinear constraints contain data to approximately that accuracy. A larger value may be appropriate if some of the problem functions are known to be of low accuracy.

Let *rowerr* be defined as the maximum nonlinear constraint violation normalized by the size of the solution. It is required to satisfy

$$rowerr = \max_i \frac{viol_i}{\|(x,s)\|} \leq r,$$

where $viol_i$ is the violation of the ith nonlinear constraint.

If $r \leq \epsilon$, the default value is used.

Major Iteration Limit i Default $= 1000$

The value of i specifies the maximum number of major iterations allowed before termination. It is intended to guard against an excessive number of linearizations of the nonlinear constraints. Setting $i = 0$ and **Major Print Level** > 0 means that the objective and constraint gradients will be checked if **Verify Level** > 0 and 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.

Major Optimality Tolerance r Default $= \sqrt{\epsilon}$
Optimality Tolerance r

This option specifies the final accuracy of the dual variables. If nag_opt_nlp1_sparse_solve (e04ug) terminates with **ifail** $= 0$, a primal and dual solution (x, s, π) will have been computed such that

$$maxgap = \max_j \frac{gap_j}{\|\pi\|} \leq r,$$

where gap_j is an estimate of the complementarity gap for the jth variable and $\|\pi\|$ is a measure of the size of the QP dual variables (or Lagrange multipliers) given by

$$\|\pi\| = \max\left(\frac{\sigma}{\sqrt{m}}, 1\right), \qquad \text{where} \qquad \sigma = \sum_{i=1}^{m} |\pi_i|.$$

It is included to make the tests independent of a scale factor on the objective function. Specifically, gap_j is computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^{\mathrm{T}} a_j$, where g_j is the jth element of the objective gradient and a_j is the associated column of the constraint matrix $\begin{pmatrix} A & -I \end{pmatrix}$:

$$gap_j = \begin{cases} d_j \min\left(x_j - l_j, 1\right) & \text{if } d_j \geq 0; \\ -d_j \min\left(u_j - x_j, 1\right) & \text{if } d_j < 0. \end{cases}$$

If $r \leq 0$, the default value is used.

Major Print Level i
Print Level

The value of i controls the amount of printout produced by the major iterations of nag_opt_nlp1_sparse_solve (e04ug), as indicated below. A detailed description of the printed output is given in Section 9.1 (summary output at each major iteration and the final solution) and Section 13 (monitoring information at each major iteration). (See also the description of **Minor Print Level**.)

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

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output (< 80 characters; see Section 9.1) for each major iteration (no printout of the final solution).
≥ 10	The final solution and one line of summary output for each major 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 (< 120 characters; see Section 13) for each major iteration (no printout of the final solution).
≥ 10	The final solution and one long line of output for each major iteration.
≥ 20	The final solution, one long line of output for each major 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), basis factorization statistics and details of the initial basis resulting from the Crash procedure (if **start** $=$ 'C'; see Section 5).

If **Major Print Level** ≥ 5 and the unit number defined by the optional parameter **Monitoring File** is the same as that defined by nag_file_set_unit_advisory (x04ab) then the summary output for each major iteration is suppressed.

Major Step Limit r Default $= 2.0$

If $r > 0, r$ limits the change in x during a linesearch. It applies to all nonlinear problems once a 'feasible solution' or 'feasible subproblem' has been found.

A linesearch determines a step α in the interval $0 < \alpha \leq \beta$, where $\beta = 1$ if there are any nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first step attempted is $\alpha_1 = \min(1, \beta)$.

In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the elements of x can lead to floating-point overflow. The parameter r is therefore used to define a step limit $\bar{\beta}$ given by

$$\bar{\beta} = \frac{r\left(1 + \|x\|_2\right)}{\|p\|_2},$$

where p is the search direction and the first evaluation of $f(x)$ is made at the (potentially) smaller step length $\alpha_1 = \min\left(1, \bar{\beta}, \beta\right)$.

Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The default value $r = 2.0$ should not affect progress on well-behaved functions, but values such as $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. If a small value of r is selected, a 'good' starting point may be required. An important application is to the class of nonlinear least squares problems.

If $r \leq 0$, the default value is used.

Minor Feasibility Tolerance r Default $= \sqrt{\epsilon}$
Feasibility Tolerance r

This option attempts to ensure that all variables eventually satisfy their upper and lower bounds to within the tolerance r. Since this includes slack variables, general linear constraints should also be satisfied to within r. Note that feasibility with respect to nonlinear constraints is judged by the value of optional parameter **Major Feasibility Tolerance** and not by r.

If the bounds and linear constraints cannot be satisfied to within r, the problem is declared *infeasible*. Let `Sinf` be the corresponding sum of infeasibilities. If `Sinf` is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

If **Scale Option** ≥ 1, feasibility is defined in terms of the *scaled* problem (since it is more likely to be meaningful).

Nonlinear functions will only be evaluated at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every effort should be made to eliminate these regions from the problem. For example, if $f(x_1, x_2) = \sqrt{x_1} + \log(x_2)$, it is essential to place lower bounds on both x_1 and x_2. If the value $r = 10^{-6}$ is used, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious; in general, you should attempt to keep x as far away from singularities as possible.)

In reality, r is used as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible and the function is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). (See also the description of **Elastic Weight**.)

If $r \leq \epsilon$, the default value is used.

Minor Iteration Limit i Default $= 500$

The value of i specifies the maximum number of iterations allowed between successive linearizations of the nonlinear constraints. A value in the range $10 \leq i \leq 50$ prevents excessive effort being expended on early major iterations, but allows later QP subproblems to be solved to completion. Note that an extra m minor iterations are allowed if the first QP subproblem to be solved starts with the all-slack basis $B = I$. (See the description of the optional parameter **Crash Option**.)

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

If $i \leq 0$, the default value is used.

Minor Optimality Tolerance r Default $= \sqrt{\epsilon}$

This option is used to judge optimality for each QP subproblem. Let the QP reduced gradients be $d_j = g_j - \pi^{\mathrm{T}} a_j$, where g_j is the jth element of the QP gradient, a_j is the associated column of the QP constraint matrix and π is the set of QP dual variables.

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

$$\frac{d_j}{\|\pi\|} \geq -r \quad \text{or} \quad \frac{d_j}{\|\pi\|} \leq r$$

respectively, and if $\dfrac{|d_j|}{\|\pi\|} \leq r$ for superbasic variables.

Note that $\|\pi\|$ is a measure of the size of the dual variables. It is included to make the tests independent of a scale factor on the objective function. (The value of $\|\pi\|$ actually used is defined in the description for optional parameter **Major Optimality Tolerance**.)

If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against r.

If $r \leq 0$, the default value is used.

Minor Print Level i Default $= 0$

The value of i controls the amount of printout produced by the minor iterations of nag_opt_nlp1_sparse_solve (e04ug) (i.e., the iterations of the quadratic programming algorithm), as indicated below. A detailed description of the printed output is given in Section 9.2 (summary output at each minor iteration) and Section 13 (monitoring information at each minor iteration). (See also the description of the optional parameter **Major Print Level**.)

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

i	**Output**
0	No output.
≥ 1	One line of summary output (< 80 characters; see Section 9.2) for each minor iteration.

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

i	**Output**
0	No output.
≥ 1	One long line of output (< 120 characters; see Section 13) for each minor iteration.

If **Minor Print Level** ≥ 1 and the unit number defined by the optional parameter **Monitoring File** is the same as that defined by nag_file_set_unit_advisory (x04ab) then the summary output for each minor iteration is suppressed.

Monitoring File i Default $= -1$

If $i \geq 0$ and **Major Print Level** ≥ 5 or $i \geq 0$ and **Minor Print Level** ≥ 1 then monitoring information is produced by nag_opt_nlp1_sparse_solve (e04ug) at every iteration is sent to a file with logical unit number i. If $i < 0$ and/or **Major Print Level** < 5 and **Minor Print Level** < 1 then no monitoring information is produced.

Partial Price i Default $= 1$ or 10

The default value of i is 1 if there are any nonlinear constraints and 10 otherwise.

This option is recommended for large 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 become superbasic). The possible choices for i are the following.

i	Meaning
1	All columns of the constraint matrix $\begin{pmatrix} A & -I \end{pmatrix}$ are searched.
≥ 2	Both A and I are partitioned to give i roughly equal segments A_j, I_j, for $j = 1, 2, \ldots, p$ (modulo 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}. 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+2}, I_{j+2} 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 during the solution of QP subproblems 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 specified search direction p, a 'ratio test' is used to determine which element of x reaches an upper or lower bound first. The corresponding element of p is called the *pivot element*. Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than r.

It is common in practice for two (or more) variables to reach a bound at essentially the same time. In such cases, the **Minor Feasibility Tolerance** provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively *small* values of **Minor Feasibility Tolerance** should therefore not be specified. To a lesser extent, the **Expand Frequency** also provides some freedom to maximize the pivot element. Excessively *large* values of **Expand Frequency** should therefore not be specified.

If $r \leq 0$, the default value is used.

Scale Option i Default $= 1$ or 2

The default value of i is 1 if there are any nonlinear constraints and 2 otherwise.

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. (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 $j > n$.) The possible choices for i are the following.

i	Meaning
0	No scaling is performed. This is recommended if it is known that the elements of x and the constraint matrix A (along with its Jacobian) never become large (say, > 1000).
1	All linear constraints and variables are scaled. This may improve the overall efficiency of the function on some problems.
2	All constraints and variables are scaled. Also, an additional scaling is performed that takes into account columns of $\begin{pmatrix} A & -I \end{pmatrix}$ that are fixed or have positive lower bounds or negative upper bounds.

If there are any nonlinear constraints present, the scale factors depend on the Jacobian at the first point that satisfies the linear constraints and the upper and lower bounds. The setting $i = 2$ should therefore be used only if a 'good' starting point is available and the problem is not highly nonlinear.

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** $= 0$.

The value r $(0 < r < 1)$ 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.

Start Objective Check At Column	i_1	Default $= 1$
Stop Objective Check At Column	i_2	Default $= n_1'$
Start Constraint Check At Column	i_3	Default $= 1$
Stop Constraint Check At Column	i_4	Default $= n_1''$

These keywords take effect only if **Verify Level** > 0. They may be used to control the verification of gradient elements computed by **objfun** and/or Jacobian elements computed by **confun**. For example, if the first 30 elements of the objective gradient appeared to be correct in an earlier run, so that only element 31 remains questionable then it is reasonable to specify **Start Objective Check At Column** $= 31$. Similarly for columns of the Jacobian. If the first 30 variables occur nonlinearly in the constraints but the remaining variables are nonlinear only in the objective, then **objfun** must set the first 30 elements of the array **objgrd** to zero, but these hardly need to be verified. Again it is reasonable to specify **Start Objective Check At Column** $= 31$.

If $i_2 \leq 0$ or $i_2 > n_1'$, the default value is used.

If $i_1 \leq 0$ or $i_1 > \min(n_1', i_2)$, the default value is used.

If $i_4 \leq 0$ or $i_4 > n_1''$, the default value is used.

If $i_3 \leq 0$ or $i_3 > \min(n_1'', i_4)$, the default value is used.

Superbasics Limit	i	Default $= \min(500, \bar{n} + 1)$

Note that this option does not apply to linear problems.

It places a limit on the storage allocated for superbasic variables. Ideally, the value of i should be set slightly larger than the 'number of degrees of freedom' expected at the solution.

For nonlinear problems, the number of degrees of freedom is often called the 'number of independent variables'. Normally, the value of i need not be greater than $\bar{n} + 1$, but for many problems it may be considerably smaller. (This will save storage if $\bar{n}$ is very large.)

If $i \leq 0$, the default value is used.

Unbounded Objective	r_1	Default $= 10^{15}$
Unbounded Step Size	r_2	Default $= \max(bigbnd, 10^{20})$

These options are intended to detect unboundedness in nonlinear problems. During the linesearch, the objective function f is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. If $|f|$ exceeds r_1 or α exceeds r_2, the iterations are terminated and the function returns with **ifail** $= 3$.

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

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

If $r_1 \leq 0$ or $r_2 \leq 0$, the appropriate default value is used.

Verify Level	i	Default $= 0$

This option refers to finite difference checks on the gradient elements computed by **objfun** and **confun**. Gradients are verified at the first point that satisfies the linear constraints and the upper and lower bounds. Unspecified gradient elements are not checked and hence they result in no overhead. The possible choices for i are the following.

i	**Meaning**
-1	No checks are performed.

0 Only a 'cheap' test will be performed, requiring three calls to **objfun** and two calls to **confun**. Note that no checks are carried out if every column of the constraint gradients (Jacobian) contains a missing element.

1 Individual objective gradient elements will be checked using a reliable (but more expensive) test. If **Major Print Level** > 0, a key of the form `OK` or `BAD?` indicates whether or not each element appears to be correct.

2 Individual columns of the constraint gradients (Jacobian) will be checked using a reliable (but more expensive) test. If **Major Print Level** > 0, a key of the form `OK` or `BAD?` indicates whether or not each element appears to be correct.

3 Check both constraint and objective gradients (in that order) as described above for $i = 2$ and $i = 1$ respectively.

The value $i = 3$ should be used whenever a new function function is being developed. The **Start Objective Check At Column** and **Stop Objective Check At Column** keywords may be used to limit the number of nonlinear variables to be checked.

If $i < -1$ or $i > 3$, the default value is used.

Violation Limit r Default $= 10.0$

This option defines an absolute limit on the magnitude of the maximum constraint violation after the linesearch. Upon completion of the linesearch, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq r \times \max(1, v_i(x_0)),$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of the violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of r. This makes it possible to keep the iterates within a region where the objective function is expected to be well-defined and bounded below (or above in the case of maximization). If the objective function is bounded below (or above in the case of maximization) for all values of the variables, then r may be any large positive value.

If $r \leq 0$, 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 nag_opt_nlp1_sparse_solve (e04ug). (See also the description of the optional parameters **Monitoring File**, **Major Print Level** and **Minor Print Level**.) You can control the level of printed output.

When **Major Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following line of intermediate printout (< 120 characters) is produced at every major 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.

`Major` is the major iteration count.

`Minor` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Minor` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 11).

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`nObj` is the number of times **objfun** has been called to evaluate the nonlinear part of the objective function. Evaluations needed for the estimation of the gradients by

finite differences are not included. `nObj` is printed as a guide to the amount of work required for the linesearch.

`nCon` is the number of times **confun** has been called to evaluate the nonlinear constraint functions (not printed if **ncnln** is zero).

`Merit` is the value of the augmented Lagrangian merit function (6) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 9.1). As the solution is approached, `Merit` will converge to the value of the objective function at the solution.

In elastic mode (see Section 11.2), the merit function is a composite function involving the constraint violations weighted by the value of the optional parameter **Elastic Weight**.

If there are no nonlinear constraints present, this entry contains `Objective`, the value of the objective function $f(x)$. In this case, $f(x)$ will decrease monotonically to its optimal value.

`Feasibl` is the value of *rowerr*, the largest element of the scaled nonlinear constraint residual vector defined in the description of the optional parameter **Major Feasibility Tolerance**. The solution is regarded as 'feasible' if `Feasibl` is less than (or equal to) the optional parameter **Major Feasibility Tolerance**. `Feasibl` will be approximately zero in the neighbourhood of a solution.

If there are no nonlinear constraints present, all iterates are feasible and this entry is not printed.

`Optimal` is the value of *maxgap*, the largest element of the maximum complementarity gap vector defined in the description of the optional parameter **Major Optimality Tolerance**. The Lagrange multipliers are regarded as 'optimal' if `Optimal` is less than (or equal to) the optional parameter **Major Optimality Tolerance**. `Optimal` will be approximately zero in the neighbourhood of a solution.

`nS` is the current number of superbasic variables.

`Penalty` is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if **ncnln** is zero).

`LU` is the number of nonzeros representing the basis factors L and U on completion of the QP subproblem.

If there are nonlinear constraints present, the basis factorization $B = LU$ is computed at the start of the first minor iteration. At this stage, `LU` $=$ `lenL` $+$ `lenU`, where `lenL` is the number of subdiagonal elements in the columns of a lower triangular matrix and `lenU` is the number of diagonal and superdiagonal elements in the rows of an upper triangular matrix. As columns of B are replaced during the minor iterations, the value of `LU` may fluctuate up or down (but in general will tend to increase). As the solution is approached and the number of minor iterations required to solve each QP subproblem decreases towards zero, `LU` will reflect the number of nonzeros in the LU factors at the start of each QP subproblem.

If there are no nonlinear constraints present, refactorization is subject only to the value of the optional parameter **Factorization Frequency** and hence `LU` will tend to increase between factorizations.

`Swp` is the number of columns of the basis matrix B that were swapped with columns of S in order to improve the condition number of B (not printed if **ncnln** is zero). The swaps are determined by an LU factorization of the rectangular matrix $B_S = \begin{pmatrix} B & S \end{pmatrix}^{\mathrm{T}}$, with stability being favoured more than sparsity.

`Cond Hz` is an estimate of the condition number of the reduced Hessian of the Lagrangian (not printed if **ncnln** and **nonln** are both zero). It is the square of the ratio between the largest and smallest diagonal elements of the upper triangular matrix

R. This constitutes a lower bound on the condition number of the matrix $R^{\mathrm{T}}R$ that approximates the reduced Hessian. The larger this number, the more difficult the problem.

`PD` is a two-letter indication of the status of the convergence tests involving the feasibility and optimality of the iterates defined in the descriptions of the optional parameters **Major Feasibility Tolerance** and **Major Optimality Tolerance**. Each letter is `T` if the test is satisfied and `F` otherwise. The tests indicate whether the values of `Feasibl` and `Optimal` are sufficiently small. For example, `TF` or `TT` is printed if there are no nonlinear constraints present (since all iterates are feasible). If either indicator is `F` when nag_opt_nlp1_sparse_solve (e04ug) terminates with **ifail** $= 0$, you should check the solution carefully.

`M` is printed if an extra evaluation of user-supplied functions **objfun** and **confun** was needed in order to define an acceptable positive definite quasi-Newton update to the Hessian of the Lagrangian. This modification is only performed when there are nonlinear constraints present.

`m` is printed if, in addition, it was also necessary to modify the update to include an augmented Lagrangian term.

`s` is printed if a self-scaled BFGS (Broyden–Fletcher–Goldfarb–Shanno) update was performed. This update is always used when the Hessian approximation is diagonal and hence always follows a Hessian reset.

`S` is printed if, in addition, it was also necessary to modify the self-scaled update in order to maintain positive-definiteness.

`n` is printed if no positive definite BFGS update could be found, in which case the approximate Hessian is unchanged from the previous iteration.

`r` is printed if the approximate Hessian was reset after 10 consecutive major iterations in which no BFGS update could be made. The diagonal elements of the approximate Hessian are retained if at least one update has been performed since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`R` is printed if the approximate Hessian has been reset by discarding all but its diagonal elements. This reset will be forced periodically by the values of the optional parameters **Hessian Frequency** and **Hessian Updates**. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`l` is printed if the change in the norm of the variables was greater than the value defined by the optional parameter **Major Step Limit**. If this output occurs frequently during later iterations, it may be worthwhile increasing the value of **Major Step Limit**.

`c` is printed if central differences have been used to compute the unknown elements of the objective and constraint gradients. A switch to central differences is made if either the linesearch gives a small step, or x is close to being optimal. In some cases, it may be necessary to re-solve the QP subproblem with the central difference gradient and Jacobian.

`u` is printed if the QP subproblem was unbounded.

`t` is printed if the minor iterations were terminated after the number of iterations specified by the value of the optional parameter **Minor Iteration Limit** was reached.

`i` is printed if the QP subproblem was infeasible when the function was not in elastic mode. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem (8) (see Section 11.2). It is also printed if the minimizer of the elastic subproblem does not satisfy the linearized constraints when the function is already in elastic mode. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`w` is printed if a weak solution of the QP subproblem was found.

When **Minor Print Level** ≥ 1 and **Monitoring File** ≥ 0, the following line of intermediate printout (< 120 characters) is produced at every minor 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.

In the description below, a 'pricing' operation is defined to be the process by which a nonbasic variable is selected to become superbasic (in addition to those already in the superbasic set). If the problem is purely linear, the variable selected will usually become basic immediately (unless it happens to reach its opposite bound and return to the nonbasic set).

`Itn` is the iteration count.

`pp` is the partial price indicator. The variable selected by the last pricing operation came from the `pp`th 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.

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

`-SBS` is the variable chosen to leave the superbasic set. It has become basic if the entry under `-B` is nonzero; otherwise it has become nonbasic.

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

`-B` is the variable removed from the basis (if any) to swap with a slack variable made superbasic by the latest pricing operation. The swap is done to ensure that there are no superbasic slacks.

`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., `+SBS` 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 $\begin{pmatrix} A & -I \end{pmatrix}$) 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 infeasibilities. This will not increase unless the iterations are in elastic mode. `Ninf` will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If x is infeasible, `Sinf` gives the value of the sum of infeasibilities at the start of the current iteration. It will usually decrease at each nonzero value of `Step`, but may occasionally increase if the value of `Ninf` decreases by a factor of 2 or more. However, in elastic mode this entry gives the value of the composite objective function (9), which will decrease monotonically at each iteration. If x is feasible, `Objective` is the value of the current QP objective function.

`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 `L + U` and rerun nag_opt_nlp1_sparse_solve (e04ug) (possibly using **start** = 'W'; see Section 5).

The following items are printed only if the problem is nonlinear or the superbasic set is non-empty (i.e., if the current solution is nonbasic).

`Norm rg` is the Euclidean norm of the reduced gradient of the QP objective function. During the optimality phase, this norm will be approximately zero after a unit step.

`nS` is the current number of superbasic variables.

`Cond Hz` is an estimate of the condition number of the reduced Hessian of the Lagrangian (not printed if **ncnln** and **nonln** are both zero). It is the square of the ratio between the largest and smallest diagonal elements of the upper triangular matrix R. This constitutes a lower bound on the condition number of the matrix $R^{\mathrm{T}}R$ that approximates the reduced Hessian. The larger this number, the more difficult the problem.

When **Major 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 before solving the next QP subproblem. Gaussian elimination is used to compute a sparse 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 = 5.0 or 100.0).

Note that B_S may be factorized at the beginning of just some of the major iterations. It is immediately followed by a factorization of B itself.

`Factorize` is the factorization count.

`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 nag_opt_nlp1_sparse_solve (e04ug) (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, `Increase` $= 100 \times$ (`lenL` + `lenU` − `Elems`)/`Elems`.

`m` is the number of rows in the problem. Note that `m` = `Ut` + `Lt` + `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.

`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(`Bmax`, `Umax`)/`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.)

`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 **Major Print Level** ≥ 20, **Monitoring File** ≥ 0 and **Crash Option** > 0 (default value $= 0$ or 3), 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 while 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 columns in the basis containing two nonzeros.

`Triangle` is the number of triangular columns in the basis with three (or more) nonzeros.

`Pad` is the number of slacks used to pad the basis (to make it a nonsingular triangle).

When **Major 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 x_j denote the jth 'column variable', for $j = 1, 2, \ldots, n$. We assume that a typical variable x_j has bounds $\alpha \leq x_j \leq \beta$.

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 x_j relative to the bounds α and β.

The various possible states are as follows:

`LL` x_j is nonbasic at its lower limit, α.

`UL` x_j is nonbasic at its upper limit, β.

`EQ` x_j is nonbasic and fixed at the value $\alpha = \beta$.

`FR` x_j is nonbasic at some value strictly between its bounds: $\alpha < x_j < \beta$.

`BS` x_j is basic. Usually $\alpha < x_j < \beta$.

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 current value, there would be no change in the value of the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the Lagrange multipliers *might* also change.

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

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

`N` *Not precisely optimal*. The variable is nonbasic. Its reduced gradient is larger than the value of the optional parameter **Major Feasibility Tolerance**.

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

`Obj Gradient` is the value of g_j at the final iterate. (If any x_j is infeasible, g_j is the gradient of the sum of infeasibilities.)

`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.

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

General linear constraints take the form $l \leq Ax \leq u$. The ith constraint is therefore of the form $\alpha \leq a_i^{\mathrm{T}} x \leq \beta$ and the value of $a_i^{\mathrm{T}} x$ 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 ith '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 $\alpha \leq F_i(x) + a_i^{\mathrm{T}}x \leq \beta$ are treated similarly, except that the row activity and degree of infeasibility are computed directly from $F_i(x) + a_i^{\mathrm{T}}x$ rather than from s_i.

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 the ith row.

`State` gives the state of the ith row relative to the bounds α and β.

The various possible states 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 `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 current value, there would be no change in the value of the objective function. The values of the basic and superbasic variables *might* change, giving a genuine alternative solution. The values of the Lagrange multipliers *might* also change.

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

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

`N` *Not precisely optimal*. The variable is nonbasic. Its reduced gradient is larger than the value of the optional parameter **Major Feasibility Tolerance**.

`Activity` is the value of $a_i^{\mathrm{T}}x$ (or $F_i(x) + a_i^{\mathrm{T}}x$ for nonlinear rows) 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 α, the lower bound specified for the ith row. `None` indicates that $\mathbf{bl}(n + i) \leq -bigbnd$.

`Upper Bound` is β, the upper bound specified for the ith row. `None` indicates that $\mathbf{bu}(n + i) \geq bigbnd$.

`Dual Activity` is the value of the dual variable π_i.

`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.