NAG Toolbox

nag_mip_iqp_sparse (h02ce)

1 Purpose

nag_mip_iqp_sparse (h02ce) obtains integer solutions to sparse linear programming and quadratic programming problems.

2 Syntax

```
[ns, xs, istate, miniz, minz, obj, clamda, ifail] = nag_mip_iqp_sparse(n, m,
iobj, ncolh, qphx, a, ha, ka, bl, bu, start, names, crname, ns, xs, intvar,
istate, strtgy, leniz, lenz, monit, 'nnz', nnz, 'nname', nname, 'lintvr',
lintvr, 'mdepth', mdepth)
[ns, xs, istate, miniz, minz, obj, clamda, ifail] = h02ce(n, m, iobj, ncolh,
qphx, a, ha, ka, bl, bu, start, names, crname, ns, xs, intvar, istate, strtgy,
leniz, lenz, monit, 'nnz', nnz, 'nname', nname, 'lintvr', lintvr, 'mdepth',
mdepth)
```

3 Description

nag_mip_iqp_sparse (h02ce) is designed to obtain integer solutions to a class of quadratic programming problems addressed by nag_opt_qpconvex1_sparse_solve (e04nk). Specifically it solves the following problem:

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

where $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ is a set of variables (some of which may be required to be integer), A is an m by n matrix and the objective function $f(x)$ may be specified in a variety of ways depending upon the particular problem to be solved. The optional parameter **Maximize** may be used to specify an alternative problem in which $f(x)$ is maximized. The possible forms for $f(x)$ are listed in Table 1, in which the prefixes LP and QP stand for 'linear programming' and 'quadratic programming' respectively, c is an n-element vector and H is the n by n second-derivative matrix $\nabla^2 f(x)$ (the *Hessian matrix*).

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

Table 1

For LP and QP problems, the unique global minimum value of $f(x)$ is found. For QP problems, you must also provide a function that computes Hx for any given vector x. (H need not be stored explicitly.)

(It is not expected that the feasibility problem of nag_opt_qpconvex1_sparse_solve (e04nk) would be relevant here.)

The function employs a 'Branch and Bound' technique to enforce the integer constraints. In this technique the problem is first solved without the integer constraints. If a variable is found to be non-integral when it is required to have an integer value then two additional problems are constructed. One bounds the variable above by the nearest integer value below the optimal value previously obtained. The second problem is formed by bounding the variable below by the nearest integer value above the optimal value. This process is continued until an integer solution is found. At this point you may elect to stop, or may continue to search for better integer solutions by examining any other sub-problems that remain to be explained.

In practice the function tries to compute an integer solution as quickly as possible using a depth-first approach, since this helps determine a realistic cut-off value. If we have a cut-off value, say the value of the function at this first integer solution, and any sub-problem, W say, has a solution value greater than this cut-off value, then subsequent sub-problems of W must have solutions greater than the value of the solution at W and therefore need not be computed. Thus a knowledge of a good cut-off value can result in fewer sub-problems being solved and thus speed up the operation of the function. (See the description of **monit** in Section 5 for details of how you can supply your own cut-off value.)

Each sub-problem is solved using nag_opt_qpconvex1_sparse_solve (e04nk). You are referred to the function document for nag_opt_qpconvex1_sparse_solve (e04nk) for details of the algorithm used.

4 References

Gill P E, Hammarling S, Murray W, Saunders M A and Wright M H (1986) Users' guide for LSSOL (Version 1.0) *Report SOL 86-1* Department of Operations Research, Stanford University

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

Gill P E, Murray W, Saunders M A and Wright M H (1986) Some theoretical properties of an augmented Lagrangian merit function *Report SOL* 86–6R 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 (1991) Inertia-controlling methods for general quadratic programming *SIAM Rev.* **33** 1–36

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

Lawson C L, Hanson R J, Kincaid D R and Krogh F T (1979) Basic linear algebra supbrograms for Fortran usage *ACM Trans. Math. Software* **5** 308–325

Murtagh B A and Saunders M A (1983) MINOS 5.0 user's guide *Report SOL 83-20* Department of Operations Research, Stanford University

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

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

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

2: **m** – INTEGER

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

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

3: **iobj** – INTEGER

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

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

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

4: **ncolh** – INTEGER

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

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

5: **qphx** – SUBROUTINE, supplied by the NAG Library or the user.

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

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

where the nonlinear variables y appear first as shown. For LP problems, **qphx** will never be called by nag_mip_iqp_sparse (h02ce).

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

Input Parameters

1: **nstate** – INTEGER

If **nstate** $= 1$, then nag_mip_iqp_sparse (h02ce) is calling **qphx** for the first time on a sub-problem. This argument setting allows you to save computation time if certain data must be read or calculated only once.

If **nstate** ≥ 2, then nag_mip_iqp_sparse (h02ce) is calling **qphx** for the last time. This argument setting allows you to perform some additional computation on the final sub-problem solution. In general, the last call to **qphx** is made with **nstate** $= 2 +$ **ifail** (see Section 6).

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

2: **ncolh** – INTEGER

This is the same argument **ncolh** as supplied to nag_mip_iqp_sparse (h02ce).

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

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

Output Parameters

1: **hx**(**ncolh**) – REAL (KIND=nag_wp) array

The product Hx.

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

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

7: **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**. Note that the row indices for a column may be supplied in any order.

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

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

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

Constraints:

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

9: **bl**$(\mathbf{n}+\mathbf{m})$ – 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, and the next **m** elements the bounds for the general linear constraints Ax (or slacks s) and the free row (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set **bl**$(j) \leq -bigbnd$, where $bigbnd$ is the value of the optional parameter **Infinite Bound Size** (default value $= 10^{20}$). To specify the jth constraint as an *equality*, set **bl**$(j) =$ **bu**$(j) = \beta$, say, where $|\beta| < bigbnd$. Note that the lower bound corresponding to the free row must be set to $-\infty$ and stored in **bl**$(\mathbf{n} +$ **iobj**$)$.

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

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

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

Constraints:

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

11: **start** – 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 matrix B.

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

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

12: **names**$(\mathbf{5})$ – CHARACTER(8) array

A set of names associated with the so-called MPSX form of the problem.

names(1)
Must contain the name for the problem (or be blank).

names(2)
Must contain the name for the free row (or be blank).

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

names(4)
Must contain the name for the ranges (or be blank).

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

(These names are used in the monitoring file output; see Section 13.)

13: **crname**(**nname**) – CHARACTER(8) array

The optional column and row names.

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

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

14: **ns** – INTEGER

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

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

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

16: **intvar**(**lintvr**) – INTEGER array

Specifies which components of the solution vector x are constrained to be integer. Specifically, if k elements of x are required to take integer values then **intvar**$(i) = l_i$, for $i = 1, 2, \ldots, k$, where l_i is the integer index such that x_{l_i} is integer. If $k <$ **lintvr** then **intvar**$(k + 1)$ must be set to -1 to signal the end of the integer variable indices.

The order in which the indices of those components of x required to be integer is presented determines the order in which the sub-problems are treated and solved. As such it can be a powerful tool to assist the function in achieving a solution efficiently. The general advice is to enter the important integer variables in the model early in **intvar**; secondary or less important variables should be entered near the end of the list. However some experimentation might be required to find the optimal order.

17: **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 columns of $(A - I)$. Possible values for **istate**(j) are as follows:

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

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

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

If **start** = 'W', **istate** and **xs** must specify the initial states and values, respectively, of the variables and slacks (x, s). If nag_mip_iqp_sparse (h02ce) 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**.

18: **strtgy** – INTEGER

Defines the branching strategy adopted by the function.

strtgy = 0
Each sub-problem first explored imposes a tighter upper bound on the component of x.

strtgy = 1
Each sub-problem first explored imposes a tighter lower bound on the component of x.

strtgy = 2
Each branch explored imposes a tighter upper bound on the component of x if its fractional part is less than 0.5, otherwise it imposes a tighter lower bound.

strtgy = 3
Random choice is made between first exploring a tighter lower bound or a tighter upper bound sub-problem.

Constraint: **strtgy** = 0, 1, 2 or 3.

19: **leniz** – INTEGER

The dimension of the array **iz**.

Constraint: **leniz** ≥ 1.

20: **lenz** – INTEGER

The dimension of the array **z**.

Constraint: **lenz** ≥ 1.

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** and **lenz** set to 1. (nag_mip_iqp_sparse (h02ce) will then terminate with **ifail** = 14.)

21: **monit** – SUBROUTINE, supplied by the NAG Library or the user.

To provide feed-back on the progress of the branch and bound process. Additionally **monit** provides, via its argument **halt**, the ability to terminate the process. (You might choose to do this when an integer solution is found, rather than search for a better solution.) If you do not require any intermediate output then **monit** may be the string `nag_mip_iqp_sparse_dummy_monit (h02cey)`.

```
[bstval, halt, count] = monit(intfnd, nodes, depth, obj, x, bstval,
bstsol, bl, bu, n, halt, count)
```

Input Parameters

1: **intfnd** – INTEGER

Contains the number of integer solutions obtained so far.

2: **nodes** – INTEGER

Contains the number of nodes (sub-problems) solved so far.

3: **depth** – INTEGER

Contains the depth reached in the tree of problems.

4: **obj** – REAL (KIND=nag_wp)

Contains the solution value to the sub-problem at this node.

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

Contains the solution vector to the sub-problem at this node.

6: **bstval** – REAL (KIND=nag_wp)

Contains the value of the objective function corresponding to the best integer solution obtained so far. If no integer solution has been found **bstval** contains the largest machine representable number (see nag_machine_real_largest (x02al)).

7: **bstsol**(**n**) – REAL (KIND=nag_wp) array

Contains the value of the best integer solution obtained so far.

8: **bl**(**n**) – REAL (KIND=nag_wp) array

Contains the current lower bounds on the variables at this point.

9: **bu**(**n**) – REAL (KIND=nag_wp) array

Contains the current upper bounds on the variables at this point.

10: **n** – INTEGER

Contains the number of variables in the minimization problem.

11: **halt** – LOGICAL

Will have the value *false*.

12: **count** – INTEGER

count may be used to save the last value of **intfnd**. If a subsequent call of **monit** has a value of **intfnd** which is greater than **count**, then you know that a new integer solution has been found at this node.

Output Parameters

1: **bstval** – REAL (KIND=nag_wp)

May be set to a cut-off value, if you are a sophisticated user, as follows. Before an integer solution has been found **bstval** will be set by nag_mip_iqp_sparse (h02ce) to the largest machine representable number (see nag_machine_real_largest (x02al)). If you

know that the solution being sought is a much smaller number, then **bstval** may be set to this number as a cut-off value (see Section 3). Beware of setting **bstval** too small, since then no integer solutions will be discovered. Also make sure that **bstval** is set using a statement of the form

`IF (intfnd.EQ.0)` **bstval** = *cut-off value*

on entry to **monit**. This statement will not prevent the normal operation of the algorithm when subsequent integer solutions are found. It would be a grievous mistake to unconditionally set **bstval** and if you have any doubts whatsoever about the correct use of this argument then you are strongly recommended to leave it unchanged.

2: **halt** – LOGICAL

If **halt** is set to *true*, nag_opt_qpconvex1_sparse_solve (e04nk) will be brought to a halt with **ifail** exit -1. This facility may be useful if you are content with *any* integer solution, or with any integer solution that fits certain criteria. Under these circumstances setting **halt** $=$ *true* can save considerable unnecessary computation.

3: **count** – INTEGER

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.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n} \times \mathbf{m}$.

2: **nname** – INTEGER

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

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

nname $= 1$
There are no names. Default names will be used in the printed output.

nname $= \mathbf{n} + \mathbf{m}$
All names must be supplied.

Constraint: **nname** $= 1$ or $\mathbf{n} + \mathbf{m}$.

3: **lintvr** – INTEGER

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

k, the number of components of x required to be integer. If $k = 0$, then **lintvr** must be set to 1 and **intvar**(1) set to -1.

4: **mdepth** – INTEGER

Suggested value: **mdepth** $= 2 \times \mathbf{n} + 20$.

Default: $2 \times \mathbf{n} + 20$

Specifies the maximum depth the tree of sub-problems may be developed.

Constraint: **mdepth** > 0.

5.3 Output Parameters

1: **ns** – INTEGER

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

2: **xs**$(\mathbf{n}+\mathbf{m})$ – REAL (KIND=nag_wp) array

xs(i) contains the final value of x_i, for $i = 1, 2, \ldots, n$.

3: **istate**$(\mathbf{n}+\mathbf{m})$ – INTEGER array

The final states of the variables and slacks (x, s) from the solution of the last sub-problem tackled. 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` $= 0$ (see Section 9.1), basic and superbasic variables may be outside their bounds by as much as the value of the optional parameter **Feasibility Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$, where ϵ is the ***machine precision***). Note that unless the optional parameter **Scale Option** $= 0$ (default value $= 2$) is specified, the **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 **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` (see Section 9.1) if **Scale Option** $= 0$).

4: **miniz** – INTEGER

The minimum value of **leniz** required to start solving the problem. If **ifail** $= 14$, nag_mip_iqp_sparse (h02ce) may be called again with **leniz** suitably larger than **miniz**. (The bigger the better, since it is not certain how much workspace the basis factors need.)

5: **minz** – INTEGER

The minimum value of **lenz** required to start solving the problem. If **ifail** $= 15$, nag_mip_iqp_sparse (h02ce) may be called again with **lenz** suitably larger than **minz**. (The bigger the better, since it is not certain how much workspace the basis factors need.)

6: **obj** – REAL (KIND=nag_wp)

The value of the objective function.

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

If `Ninf` > 0, **obj** is just the linear objective term $c^{\mathrm{T}}x$ (if any). For FP problems, **obj** is set to zero.

7: **clamda**$(\mathbf{n}+\mathbf{m})$ – REAL (KIND=nag_wp) array

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

8: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= -1$ (*warning*)

Halted at your request.

ifail $= 0$

Successful exit.

ifail $= 1$

Input argument error immediately detected.

ifail $= 2$

No integer solution found.

ifail $= 3$

mdepth is too small.

ifail $= 4$

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

ifail $= 5$

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

ifail $= 6$

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

ifail $= 7$

The reduced Hessian matrix $z^{\mathrm{T}}HZ$ (see Section 11.2) exceeds its assigned dimension. The value of the optional parameter **Superbasics Limit** (default value $= \min(n_H + 1, n)$) is too small.

ifail $= 8$

The Hessian matrix H appears to be indefinite. Check that **qphx** has been coded correctly and that all relevant elements of Hx have been assigned their correct values.

ifail $= 9$

An input argument is invalid for an internal call to nag_opt_qpconvex1_sparse_solve (e04nk).

ifail $= 10$

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

ifail $= 11$

Not enough integer workspace for the basis factors. Increase **leniz** and rerun nag_mip_iqp_sparse (h02ce).

ifail $= 12$

Not enough real workspace for the basis factors. Increase **lenz** and rerun nag_mip_iqp_sparse (h02ce).

ifail $= 13$

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

ifail $= 14$

Not enough integer workspace to start solving the problem. Increase **leniz** to at least **miniz** and rerun nag_mip_iqp_sparse (h02ce).

ifail $= 15$

Not enough real workspace to start solving the problem. Increase **lenz** to at least **minz** and rerun nag_mip_iqp_sparse (h02ce).

ifail $= 16$

An internal error has occurred. Contact NAG with details of your program.

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

nag_mip_iqp_sparse (h02ce) implements a numerically stable active-set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

8 Further Comments

This section contains a description of the printed output.

8.1 Description of the Printed Output

This section describes the (default) intermediate printout and final printout produced by nag_mip_iqp_sparse (h02ce). 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 **Print Level** in Section 12.1). Note that the intermediate printout and final printout are produced only if **Print Level** ≥ 10 (the default).

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

`Itn` is the iteration count.

`Step` is the step taken along the computed search direction. If a constraint is added during the current iteration, `Step` will be the step to the nearest constraint. When the problem is of type LP, the step can be greater than one during the optimality phase.

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

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

During the optimality phase, the value of the objective function will be nonincreasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Norm rg` is $\|d_S\|$, the Euclidean norm of the reduced gradient (see Section 11.3). During the optimality phase, this norm will be approximately zero after a unit step. For FP and LP problems, `Norm rg` is not printed.

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

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

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

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

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

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

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

`I` *Infeasible*. The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the optional parameter **Feasibility Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$, where ϵ is the ***machine precision***).

`N` *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional

parameter **Optimality Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$), the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

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

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

`Constrnt` gives the name of the linear constraint.

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

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

9 Example

This example minimizes the quadratic function $f(x) = c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$, where

$$c = (-200.0, -2000.0, -2000.0, -2000.0, -2000.0, 400.0, 400.0)^{\mathrm{T}}$$

$$H = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 \end{pmatrix}$$

subject to the bounds

$$\begin{aligned} 0 &\leq x_1 \leq 200 \\ 0 &\leq x_2 \leq 2500 \\ 400 &\leq x_3 \leq 800 \\ 100 &\leq x_4 \leq 700 \\ 0 &\leq x_5 \leq 1500 \\ 0 &\leq x_6 \\ 0 &\leq x_7 \end{aligned}$$

to the linear constraints

$$\begin{array}{rcrcrcrcrcrcrcr} & & x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & + & x_6 & + & x_7 & = & 2000 \\ & & 0.15x_1 & + & 0.04x_2 & + & 0.02x_3 & + & 0.04x_4 & + & 0.02x_5 & + & 0.01x_6 & + & 0.03x_7 & \leq & 60 \\ & & 0.03x_1 & + & 0.05x_2 & + & 0.08x_3 & + & 0.02x_4 & + & 0.06x_5 & + & 0.01x_6 & & & \leq & 100 \\ & & 0.02x_1 & + & 0.04x_2 & + & 0.01x_3 & + & 0.02x_4 & + & 0.02x_5 & & & & & \leq & 40 \\ & & 0.02x_1 & + & 0.03x_2 & & & & & + & 0.01x_5 & & & & & \leq & 30 \\ 1500 & \leq & 0.70x_1 & + & 0.75x_2 & + & 0.80x_3 & + & 0.75x_4 & + & 0.80x_5 & + & 0.97x_6 & & & & \\ 250 & \leq & 0.02x_1 & + & 0.06x_2 & + & 0.08x_3 & + & 0.12x_4 & + & 0.02x_5 & + & 0.01x_6 & + & 0.97x_7 & \leq & 300 \end{array}$$

and the variables x_2, x_3, x_4, x_5, x_6, x_7, are constrained to be integer.

The initial point, which is infeasible, is

$$x_0 = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)^{\mathrm{T}}.$$

The optimal solution (to five figures) is

$$x^* = (0.0, 355.0, 645.0, 164.0, 410.0, 275.0, 151.0)^{\mathrm{T}}.$$

One bound constraint and one linear constraint are active at the solution. Note that the Hessian matrix H is positive semidefinite.

9.1 Program Text

```
     function h02ce_example

fprintf('h02ce example results\n\n');

n     = nag_int(7);
m     = nag_int(8);
iobj  = m;
ncolh = n;
big   = 1.e25;

a = [ 1.00  0.15  0.03  0.02  0.02  0.70  0.02   -200 ...
      1.00  0.04  0.05  0.04  0.03  0.75  0.06  -2000 ...
      1.00  0.02  0.08  0.01        0.80  0.08  -2000 ...
      1.00  0.04  0.02  0.02        0.75  0.12  -2000 ...
      1.00  0.02  0.06  0.02  0.01  0.80  0.02  -2000 ...
      1.00  0.01  0.01              0.97  0.01    400 ...
      1.00  0.03                          0.97    400];
ha = nag_int([1 2 3 4 5 6 7 8 ...
              1 2 3 4 5 6 7 8 ...
              1 2 3 4   6 7 8 ...
              1 2 3 4   6 7 8 ...
              1 2 3 4 5 6 7 8 ...
              1 2 3     6 7 8  ...
              1 2         7 8]);
ka = [nag_int(1) 9 17 24 31 39 45 49];
bl = [   0      0    400    100      0     0    0  ...
      2000   -big   -big   -big   -big  1500  250 -big];
bu = [ 200   2500    800    700   1500   big  big  ...
      2000     60    100     40     30   big  300  big];
start = 'C';
names = {'        ', '        ', '        ', '        ', '        '};
crname = {'...x1...', '...x2...', '...x3...', '...x4...', '...x5...'  ...
          '...x6...', '...x7...', '..row1..', '..row2..', '..row3..'  ...
          '..row4..', '..row5..', '..row6..', '..row7..', '..cost..'};
ns = nag_int(0);
xs = zeros(n+m,1);
intvar = [nag_int(2) 3 4 5 6 7 -1 0 0 0];
istate = zeros(n+m, 1, nag_int_name);
strtgy = nag_int(3);
leniz  = nag_int(100000);
lenz   = nag_int(100000);

% Print Options
h02cg('Nolist');
h02cg('Print level = 0');

[ns, xs, istate, miniz, minz, obj, clamda, ifail] = ...
h02ce( ...
       n, m, iobj, ncolh, @qphx, a, ha, ka, bl, bu, start, names, ...
       crname, ns, xs, intvar, istate, strtgy, leniz, lenz, @monit);

fprintf('Optimal Integer Value is = %20.8e\n',obj);
disp('Components are:');
for j=1:7
  fprintf('x(%2d) = %12.8f\n',j,xs(j));
end
```

```
function [hx] = qphx(nstate, ncolh, x)
  hx = zeros(ncolh,1);
  hx(1) = 2*x(1);
  hx(2) = 2*x(2);
  hx(3) = 2*(x(3)+x(4));
  hx(4) = hx(3);
  hx(5) = 2*x(5);
  hx(6) = 2*(x(6)+x(7));
  hx(7) = hx(6);

function [bstval, halt, count] = monit(intfnd,nodes,depth,obj,x,bstval, ...
                                       bstsol,bl,bu,n,halt,count)
                                      halt = false;
  if intfnd == 0
    bstval = -1847510;
  elseif intfnd>count
    fprintf('New integer solution found\n');
    fprintf('  Nodes solved so far: %20d\n', nodes);
    fprintf('  Reached depth:       %20d\n', depth);
    fprintf('  Solution value at current node: %13.5e\n', obj);
    fprintf('  Solution vector at current node:\n');
    fprintf('     %13.5e\n',x);
    fprintf('  Current best function value: %13.5e\n', bstval);
    fprintf('  Current best solution:\n');
    fprintf('     %13.5e\n',bstsol);
    fprintf('  Current lower and upper bounds:\n');
    fprintf('    %13.5e    %13.5e\n', [bl' bu']');
    fprintf('\n');
  end
  count = intfnd;
```

9.2 Program Results

```
     h02ce example results

New integer solution found
  Nodes solved so far:                  272
  Reached depth:                         18
  Solution value at current node:  -1.84752e+06
  Solution vector at current node:
       0.00000e+00
       3.55000e+02
       6.45000e+02
       1.64000e+02
       4.10000e+02
       2.75000e+02
       1.51000e+02
  Current best function value:  -1.84752e+06
  Current best solution:
       0.00000e+00
       3.55000e+02
       6.45000e+02
       1.64000e+02
       4.10000e+02
       2.75000e+02
       1.51000e+02
  Current lower and upper bounds:
      0.00000e+00      2.00000e+02
      3.55000e+02      3.55000e+02
      4.00000e+02      6.45000e+02
      1.64000e+02      1.64000e+02
      0.00000e+00      1.50000e+03
      0.00000e+00      1.00000e+25
      0.00000e+00      1.00000e+25

Optimal Integer Value is =      -1.84751800e+06
Components are:
x( 1) =   0.00000000
x( 2) = 355.00000000
```

```
x( 3) = 645.00000000
x( 4) = 164.00000000
x( 5) = 410.00000000
x( 6) = 275.00000000
x( 7) = 151.00000000
```

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_mip_iqp_sparse_optstr (h02cg). 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_mip_iqp_sparse (h02ce).

10.1 Overview

nag_mip_iqp_sparse (h02ce) employs a Branch and Bound technique (see Section 3) based on an inertia-controlling method to solve the sub-problems that maintains a Cholesky factorization of the reduced Hessian (see below). The method is similar to that of Gill and Murray (1978), and is described in detail by Gill *et al.* (1991). Here we briefly summarise the main features of the method. Where possible, explicit reference is made to the names of variables that are arguments of the function or appear in the printed output.

The method used has two distinct phases: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same functions. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities (the printed quantity `Sinf`; see Section 13) to the quadratic objective function (the printed quantity `Objective`; see Section 13).

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

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

where the *step length* α is a non-negative scalar (the printed quantity `Step`; see Section 13), and p is called the *search direction*. (For simplicity, we shall consider a typical iteration and avoid reference to the index of the iteration.) Once an iterate is feasible (i.e., satisfies the constraints), all subsequent iterates remain feasible.

10.2 Definition of the Working Set and Search Direction

At each iterate (x, s), a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the value of the optional parameter **Feasibility Tolerance**; see Section 12.1). The working set is the current prediction of the constraints that hold with equality at a solution of the LP or QP problem. Let m_W denote the number of constraints in the working set (including bounds), and let W denote the associated m_W by $(n + m)$ *working set matrix* consisting of the m_W gradients of the working set constraints.

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

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

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

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

where p_z is any n_z-vector.

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

The working set matrix W can be represented in terms of a certain column partition of the matrix $(A - I)$. As in Section 3 we partition the constraints $Ax - s = 0$ so that

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

where B is a square nonsingular basis and x_B, x_S and $x_{\mathbf{n}}$ are the basic, superbasic and nonbasic variables respectively. The nonbasic variables are equal to their upper or lower bounds at (x, s), and the superbasic variables are independent variables that are chosen to improve the value of the current objective function. The number of superbasic variables is n_S (the printed quantity `Ns`; see Section 13). Given values of x_N and x_S, the basic variables x_B are adjusted so that (x, s) satisfies (5).

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

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

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

The null space matrix Z is defined from a sparse LU factorization of part of W. In particular, z is maintained in 'reduced gradient' form, using the LUSOL package (see Gill *et al.* (1986)) to maintain sparse LU factors of the basis matrix B that alters as the working set W changes. Given the permutation P, the null space basis is given by

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

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

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

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

where g is the objective gradient at (x, s). Roughly speaking, g_z and H_z describe the first and second derivatives of an n_S-dimensional *unconstrained* problem for the calculation of p_Z. (The condition estimator of H_Z is the quantity `Cond Hz` in the monitoring file output; see Section 13.)

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

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

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

10.3 The Main Iteration

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

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

A Lagrange-multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\sigma$ when the associated constraint is at its *lower bound*, where σ depends on the value of the optional parameter **Optimality Tolerance** (see Section 12.1). If a multiplier is nonoptimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by continuing the minimization with the corresponding constraint excluded from the working set. (This step is sometimes referred to as 'deleting' a constraint from the working set.) If optimal multipliers occur during the feasibility phase but the sum of infeasibilities is nonzero, there is no feasible point and the function terminates immediately with **ifail** $= 3$ (see Section 6).

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

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

where π satisfies the equations $B^{\mathrm{T}}\pi = g_B$, and g_B denotes the basic elements of g. The elements of π are the Lagrange-multipliers λ_j associated with the equality constraints $Ax - s = 0$. The vector d_N of nonbasic elements of d consists of the Lagrange-multipliers λ_j associated with the upper and lower bound constraints in the working set. The vector d_S of superbasic elements of d is the reduced gradient g_z in (8). The vector d_B of basic elements of d is zero, by construction. (The Euclidean norm of d_S and the final values of d_S, g and π are the quantities `Norm rg`, `Reduced Gradnt`, `Obj Gradient` and `Dual Activity` in the monitoring file output; see Section 13.)

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

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

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

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

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

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

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

10.4 Miscellaneous

If the basis matrix is not chosen carefully, the condition of the null space matrix Z in (7) could be arbitrarily high. To guard against this, the function implements a 'basis repair' feature in which the LUSOL package (see Gill *et al.* (1986)) is used to compute the rectangular factorization

$$\begin{pmatrix} B & S \end{pmatrix}^{\mathrm{T}} = LU, \tag{13}$$

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

The EXPAND procedure (see Gill *et al.* (1989)) is used to reduce the possibility of cycling at a point where the active constraints are nearly linearly dependent. Although there is no absolute guarantee that cycling will not occur, the probability of cycling is extremely small (see Gill *et al.* (1986)). The main feature of EXPAND is that the feasibility tolerance is increased at the start of every iteration. This allows a positive step to be taken at every iteration, perhaps at the expense of violating the bounds on (x, s) by a small amount.

Suppose that the value of the optional parameter **Feasibility Tolerance** is δ. Over a period of K iterations (where K is the value of the optional parameter **Expand Frequency**; see Section 12.1), the feasibility tolerance actually used by nag_mip_iqp_sparse (h02ce) (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/K$).

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

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

The resetting procedure is also invoked when nag_mip_iqp_sparse (h02ce) reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any nontrivial adjustments are made, iterations are continued.

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

11 Optional Parameters

Several optional parameters in nag_mip_iqp_sparse (h02ce) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_mip_iqp_sparse (h02ce) 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.

Check Frequency

Crash Option

Crash Tolerance

Defaults

Expand Frequency

Factorization Frequency
Feasibility Tolerance
Infinite Bound Size
Infinite Step Size
Iteration Limit
Iters
Itns
List
LU Factor Tolerance
LU Singularity Tolerance
LU Update Tolerance
Maximize
Minimize
Monitoring File
Nolist
Optimality Tolerance
Partial Price
Pivot Tolerance
Print Level
Rank Tolerance
Scale Option
Scale Tolerance
Superbasics Limit

Optional parameters may be specified by calling nag_mip_iqp_sparse_optstr (h02cg) prior to a call to nag_mip_iqp_sparse (h02ce).

nag_mip_iqp_sparse_optstr (h02cg) can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
h02cg('Print Level = 5')
```

nag_mip_iqp_sparse_optstr (h02cg) 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_mip_iqp_sparse (h02ce) (unless they define invalid values) and so remain in effect for subsequent calls 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.

Check Frequency i Default $= 60$

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

Crash Option i Default $= 2$

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

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

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

Crash Tolerance r Default $= 0.1$

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

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

Defaults

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

Expand Frequency i Default $= 10000$

This option is part of an anti-cycling procedure (see Section 11.4) designed to allow progress even on highly degenerate problems.

For LP problems, the strategy is to force a positive step at every iteration, at the expense of violating the constraints by a small amount. Suppose that the value of the optional parameter **Feasibility Tolerance** is δ. Over a period of i iterations, the feasibility tolerance actually used by nag_mip_iqp_sparse (h02ce) (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/i$).

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

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

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

Factorization Frequency i Default $= 100$

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

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

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

nag_mip_iqp_sparse (h02ce) attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is assumed to be *infeasible*. Let `Sinf` be the corresponding sum of infeasibilities. If `Sinf` is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected. Note that the function does *not* attempt to find the minimum value of `Sinf`.

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

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

If $r > 0$, r defines the 'infinite' bound $bigbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $bigbnd$ will be regarded as $+\infty$ (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.

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

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

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

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

List Default
Nolist

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

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

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

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

where the multipliers μ will satisfy $|\mu| \leq r_i$. The default values of r_1 and r_2 usually strike a good compromise between stability and sparsity. For large and relatively dense problems, setting r_1 and r_2 to 25 (say) may give a marked improvement in sparsity without impairing stability to a serious degree. Note that for band matrices it may be necessary to set r_1 in the range $1 \leq r_1 < 2$ in order to achieve stability. If $r_1 < 1$ or $r_2 < 1$, the default value is used.

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

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

Minimize Default
Maximize

This option specifies the required direction of the optimization. It applies to both linear and nonlinear terms (if any) in the objective function. Note that if two problems are the same except that one minimizes $f(x)$ and the other maximizes $-f(x)$, their solutions will be the same but the signs of the dual variables π_i and the reduced gradients d_j (see Section 11.3) will be reversed.

Monitoring File i Default $= -1$

If $i \geq 0$ and **Print Level** > 0, monitoring information produced by nag_mip_iqp_sparse (h02ce) is sent to a file with logical unit number i. If $i < 0$ and/or **Print Level** $= 0$, the default value is used and hence no monitoring information is produced.

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

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

Partial Price i Default $= 10$

Note that this option does not apply to QP problems.

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

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

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

Print Level i Default $= 10$

The value of i controls the amount of printout produced by nag_mip_iqp_sparse (h02ce), as indicated below. A detailed description of the printed output is given in Section 9.1 (summary output at each iteration and the final solution) and Section 13 (monitoring information at each iteration). Note that the summary output will not exceed 80 characters per line and that the monitoring information will not exceed 120 characters per line. If $i < 0$, the default value is used. The following printout is sent to the current advisory message unit (as defined by nag_file_set_unit_advisory (x04ab)):

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

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

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

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

Rank Tolerance r Default $= 100\epsilon$

Scale Option i Default $= 2$

This option enables you to scale the variables and constraints using an iterative procedure due to Fourer (see Hock and Schittkowski (1981)), which attempts to compute row scales r_i and column scales c_j such that the scaled matrix coefficients $\bar{a}_{ij} = a_{ij} \times \left(c_j/r_i\right)$ are as close as possible to unity. This may improve the overall efficiency of the function on some problems. (The lower and upper bounds on the variables and slacks for the scaled problem are redefined as $\bar{l}_j = l_j/c_j$ and $\bar{u}_j = u_j/c_j$ respectively, where $c_j \equiv r_{j-n}$ if $j > n$.)

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

Scale Tolerance r Default $= 0.9$

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

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

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

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

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

12 Description of Monitoring Information

This section describes the intermediate printout and final printout which constitutes the monitoring information produced by nag_mip_iqp_sparse (h02ce). (See also the description of the optional parameters **Monitoring File** and **Print Level** in Section 12.1.) You can control the level of printed output.

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

`Itn` is the iteration count.

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

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

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

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

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

`-B` is the variable chosen to leave the set of basics (if any) in a special basic $\leftrightarrow$ superbasic swap. The entry under `-S` has become basic if this entry is nonzero, and nonbasic otherwise. The swap is done to ensure that there are no superbasic slacks.

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

`Pivot` is the rth element of a vector y satisfying $By = a_q$ whenever a_q (the qth column of the constraint matrix $(A - I)$) replaces the rth column of the basis matrix B. Wherever possible, `Step` is chosen so as to avoid extremely small values of `Pivot` (since they may cause the basis to be nearly singular). In extreme cases, it may be necessary to increase the value of the optional parameter **Pivot Tolerance** (default value $= \epsilon^{0.67}$, where ϵ is the ***machine precision***) to exclude very small elements of y from consideration during the computation of `Step`.

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

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

During the optimality phase, the value of the objective function will be nonincreasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`L` is the number of nonzeros in the basis factor L. Immediately after a basis factorization $B = LU$, this is `lenL`, the number of subdiagonal elements in the

columns of a lower triangular matrix. Further nonzeros are added to `L` when various columns of B are later replaced. (Thus, `L` increases monotonically.)

`U` is the number of nonzeros in the basis factor U. Immediately after a basis factorization, this is `lenU`, the number of diagonal and superdiagonal elements in the rows of an upper triangular matrix. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of `U` may fluctuate up or down; in general, it will tend to increase.

`Ncp` 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 **lenz** by at least `L + U` and rerun nag_mip_iqp_sparse (h02ce) (possibly using **start** = 'W'; see Section 5).

`Norm rg` is $\|d_S\|$, the Euclidean norm of the reduced gradient (see Section 11.3). During the optimality phase, this norm will be approximately zero after a unit step. For FP and LP problems, `Norm rg` is not printed.

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

`Cond Hz` is a lower bound on the condition number of the reduced Hessian (see Section 11.2). The larger this number, the more difficult the problem. For FP and LP problems, `Cond Hz` is not printed.

When **Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by **Monitoring File** whenever the matrix B or $B_S = \begin{pmatrix} B & S \end{pmatrix}^{\mathrm{T}}$ is factorized. Gaussian elimination is used to compute an LU factorization of B or B_S, where PLP^{T} is a lower triangular matrix and PUQ is an upper triangular matrix for some permutation matrices P and Q. The factorization is stabilized in the manner described under the **LU Factor Tolerance** (default value = 100.0; see Section 12.1).

`Factorize` is the factorization count.

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

Code	Meaning
0	First LU factorization.
1	Number of updates reached the value of the optional parameter **Factorization Frequency** (default value = 100).
2	Excessive nonzeros in updated factors.
7	Not enough storage to update factors.
10	Row residuals too large (see the description for the optional parameter **Check Frequency**).
11	Ill-conditioning has caused inconsistent results.

`Iteration` is the iteration count.

`Nonlinear` is the number of nonlinear variables in 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_mip_iqp_sparse (h02ce) (possibly using **start** = 'W'; see Section 5).

`Merit` is the average Markowitz merit count for the elements chosen to be the diagonals of PUQ. Each merit count is defined to be $(c-1)(r-1)$, where c and r are the number of nonzeros in the column and row containing the element at the time it is selected to be the next diagonal. `Merit` is the average of `m` such quantities. It gives an indication of how much work was required to preserve sparsity during the factorization.

`lenL` is the number of nonzeros in L.

`lenU` is the number of nonzeros in U.

`Increase` is the percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B. More precisely, $\texttt{Increase} = 100 \times (\texttt{lenL} + \texttt{lenU} - \texttt{Elems})/\texttt{Elems}$.

`m` is the number of rows in the problem. Note that $\texttt{m} = \texttt{Ut} + \texttt{Lt} + \texttt{bp}$.

`Ut` is the number of triangular rows of B at the top of U.

`d1` is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.

`Lmax` is the maximum subdiagonal element in the columns of L (not printed if B_S is factorized). This will not exceed the value of the **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, *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 **LU Factor Tolerance** to a value near 1.0.

`Umax` is not printed if B_S is factorized.

`Umin` is the magnitude of the smallest diagonal element of PUQ (not printed if B_S is factorized).

`Growth` is the value of the ratio `Umax`/`Bmax`, which should not be too large.

Providing `Lmax` is not large (say < 10.0), the ratio $\max(\texttt{Bmax}, \texttt{Umax})/\texttt{Umin}$ is an estimate of the condition number of B. If this number is extremely large, the basis is nearly singular and some numerical difficulties could occur in subsequent computations. (However, an effort is made to avoid near singularity by using slacks to replace columns of B that would have made `Umin` extremely small, and the modified basis is refactorized.)

`Growth` is not printed if B_S is factorized.

`Lt` is the number of triangular columns of B at the beginning of L.

`bp` is the size of the 'bump' or block to be factorized nontrivially after the triangular rows and columns have been removed.

`d2` is the number of columns remaining when the density of the basis matrix being factorized reached 0.6.

When **Print Level** ≥ 20 and **Monitoring File** ≥ 0, the following lines of intermediate printout (< 80 characters) are produced on the unit number specified by **Monitoring File** whenever **start** = 'C' (see

Section 5). They refer to the number of columns selected by the crash procedure during each of several passes through A, whilst searching for a triangular basis matrix.

`Slacks` is the number of slacks selected initially.

`Free cols` is the number of free columns in the basis.

`Preferred` is the number of 'preferred' columns in the basis (i.e., **istate**$(j) = 3$ for some $j \leq n$).

`Unit` is the number of unit columns in the basis.

`Double` is the number of double columns in the basis.

`Triangle` is the number of triangular columns in the basis.

`Pad` is the number of slacks used to pad the basis.

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

`Name` gives the name for the problem (blank if none).

`Objective` gives the name of the free row for the problem (blank if none).

`RHS` gives the name of the constraint right-hand side for the problem (blank if none).

`Ranges` gives the name of the ranges for the problem (blank if none).

`Bounds` gives the name of the bounds for the problem (blank if none).

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

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

`Number` is the column number j. (This is used internally to refer to x_j in the intermediate output.)

`Column` gives the name of x_j.

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

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

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

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

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

(default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$, where ϵ is the ***machine precision***).

`N` *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Optimality Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$), the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

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

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

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

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

`Reduced Gradnt` is the value of d_j at the final iterate (see Section 11.3). For FP problems, d_j is set to zero.

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

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

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

`Row` gives the name of ν_i.

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

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

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

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

`I` *Infeasible*. The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the optional parameter **Feasibility Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$, where ϵ is the ***machine precision***).

`N` *Not precisely optimal*. The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Optimality Tolerance** (default value $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$), the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

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

`Slack Activity`	is the value by which v_i differs from its nearest bound. (For the free row (if any), it is set to `Activity`.)
`Lower Bound`	is the lower bound specified for the variable. `None` indicates that $\mathbf{bl}(j) \leq -bigbnd$.
`Upper Bound`	is the upper bound specified for the variable. `None` indicates that $\mathbf{bu}(j) \geq bigbnd$.
`i`	gives the index i of v_i.

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