NAG C Library Function Document

nag_opt_sparse_convex_qp_solve (e04nqc)

Note: *this function uses* **optional arguments** *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional arguments, you need only read Sections 1 to 9 of this document. Refer to the additional Sections 10, 11 and 12 for a detailed description of the algorithm, the specification of the optional arguments and a description of the monitoring information produced by the function.*

1 Purpose

nag_opt_sparse_convex_qp_solve (e04nqc) solves sparse linear programming or convex quadratic programming problems. The initialization function nag_opt_sparse_convex_qp_init (e04npc) **must** have been called prior to calling nag_opt_sparse_convex_qp_solve (e04nqc).

2 Specification

```
#include <nag.h>
#include <nage04.h>
```

```
void nag_opt_sparse_convex_qp_solve (Nag_Start start,
     void (*qphx)(Integer ncolh, const double x[], double hx[], Integer nstate,
          Nag_Comm *comm),
     Integer m, Integer n, Integer ne, Integer nname, Integer lenc, Integer ncolh,
     Integer iobj, double objadd, const char *prob, const double acol[],
     const Integer inda[], const Integer loca[], const double bl[],
     const double bu[], const double c[], const char *names[],
     const Integer helast[], Integer hs[], double x[], double pi[], double rc[],
     Integer *ns, Integer *ninf, double *sinf, double *obj, Nag_E04State *state,
     Nag_Comm *comm, NagError *fail)
```

Before calling nag_opt_sparse_convex_qp_solve (e04nqc) or one of the option setting functions nag_opt_sparse_convex_qp_option_set_file (e04nrc), nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) or nag_opt_sparse_convex_qp_option_set_double (e04nuc), nag_opt_sparse_convex_qp_init (e04npc) **must** be called. The specification for nag_opt_sparse_convex_qp_init (e04npc) is:

```
void nag_opt_sparse_convex_qp_init (Nag_E04State *state, NagError *fail)
```

After calling nag_opt_sparse_convex_qp_solve (e04nqc) you can call one or both of the functions nag_opt_sparse_convex_qp_option_get_integer (e04nxc) or nag_opt_sparse_convex_qp_option_get_double (e04nyc) to obtain the current value of an optional argument.

3 Description

nag_opt_sparse_convex_qp_solve (e04nqc) is designed to solve large scale *linear* or *quadratic programming problems* that are assumed to be stated in the following general form:

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

where x is a set of n variables, l and u are constant lower and upper bounds, and A is a sparse matrix and $f(x)$ is a linear or quadratic objective function that may be specified in a variety of ways, depending upon the particular problem being solved. The option **Maximize** (see Section 11.2) may be used to specify a problem in which $f(x)$ is maximized instead of minimized.

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 may be defined as an equality by setting $l_j = u_j$. If certain bounds are not present, the associated elements of l or u may be set to special values that are treated as $-\infty$ or $+\infty$.

The possible forms for the function $f(x)$ are summarized in Table 1. The most general form for $f(x)$ is

$$f(x) = q + c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx = q + \sum_{j=1}^{n} c_j x_j + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i H_{ij} x_j$$

where q is a constant, c is a constant n vector and H is a constant symmetric n by n matrix with elements $\{H_{ij}\}$. In this form, f is a quadratic function of x and (1) is known as a *quadratic program* (QP). nag_opt_sparse_convex_qp_solve (e04nqc) is suitable for all *convex* quadratic programs. The defining feature of a *convex* QP is that the matrix H must be *positive semi-definite*, i.e., it must satisfy $x^{\mathrm{T}}Hx \geq 0$ for all x. If not, $f(x)$ is nonconvex and nag_opt_sparse_convex_qp_solve (e04nqc) will terminate with the error indicator **fail.code** = **NE_HESS_INDEF**. If $f(x)$ is nonconvex it may be more appropriate to call nag_opt_sparse_nlp_solve (e04vhc) instead.

Problem type	**Objective function $f(x)$**	**Hessian matrix H**
FP	Not applicable	$q = c = H = 0$
LP	$q + c^{\mathrm{T}}x$	$H = 0$
QP	$q + c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$	Symmetric positive semi-definite

Table 1
Choices for the objective function $f(x)$

If $H = 0$, then $f(x) = q + c^{\mathrm{T}}x$ and the problem is known as a *linear program* (LP). In this case, rather than defining an H with zero elements, you can define H to have no columns by setting **ncolh** $= 0$ (see Section 5).

If $H = 0$, $q = 0$, and $c = 0$, there is no objective function and the problem is a *feasible point problem* (FP), which is equivalent to finding a point that satisfies the constraints on x. In the situation where no feasible point exists, several options are available for finding a point that minimizes the constraint violations (see the option **Elastic Mode** in Section 11.2).

nag_opt_sparse_convex_qp_solve (e04nqc) is suitable for large LPs and QPs in which the matrix A is *sparse*, i.e., when there are sufficiently many zero elements in A to justify storing them implicitly. The matrix A is input to nag_opt_sparse_convex_qp_solve (e04nqc) by means of the three array arguments **acol**, **inda** and **loca**. This allows you to specify the pattern of non-zero elements in A.

nag_opt_sparse_convex_qp_solve (e04nqc) exploits structure or sparsity in H by requiring H to be defined *implicitly* in a function that computes the product Hx for any given vector x. In many cases, the product Hx can be computed very efficiently for any given x, e.g., H may be a sparse matrix, or a sum of matrices of rank-one.

For problems in which A can be treated as a *dense* matrix, it is usually more efficient to use nag_opt_lp (e04mfc), nag_opt_lin_lsq (e04ncc) or nag_opt_qp (e04nfc).

There is considerable flexibility allowed in the definition of $f(x)$ in Table 1. The vector c defining the linear term $c^{\mathrm{T}}x$ can be input in three ways: as a sparse row of A; as an explicit dense vector c; or as both a sparse row and an explicit vector (in which case, $c^{\mathrm{T}}x$ will be the sum of two linear terms). When stored in A, c is the **iobj**th row of A, which is known as the *objective row*. The objective row must always be a *free* row of A in the sense that its lower and upper bounds must be $-\infty$ and $+\infty$. Storing c as part of A is recommended if c is a sparse vector. Storing c as an explicit vector is recommended for a sequence of problems, each with a different objective (see arguments **c** and **lenc**).

The upper and lower bounds on the m elements of Ax are said to define the *general constraints* of the problem. Internally, nag_opt_sparse_convex_qp_solve (e04nqc) converts the general constraints to equalities by introducing a set of *slack variables* s, where $s = (s_1, s_2, \ldots, s_m)^{\mathrm{T}}$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$, together with the bounded slack

$5 \leq s_1 \leq +\infty$. The problem defined by (1) can therefore be re-written in the following equivalent form:

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

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

Each LP or QP problem is solved using an *active-set* method. This is an iterative procedure with two phases: a *feasibility phase* (*Phase 1*), in which the sum of infeasibilities is minimized to find a feasible point; and an *optimality phase* (*Phase 2*), in which $f(x)$ is minimized (or maximized) by constructing a sequence of iterations that lies within the feasible region.

Phase 1 involves solving a linear program of the form

Phase 1

$$\underset{x,v,w}{\text{minimize}} \sum_{j=1}^{n+m} (v_j + w_j) \quad \text{subject to } Ax - s = 0, \quad \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} - v + w \leq u, \quad v \geq 0, \quad w \geq 0$$

which is equivalent to minimizing the sum of the constraint violations. If the constraints are feasible (i.e., at least one feasible point exists), eventually a point will be found at which both v and w are zero. The associated value of (x, s) satisfies the original constraints and is used as the starting point for the Phase 2 iterations for minimizing $f(x)$.

If the constraints are infeasible (i.e., $v \neq 0$ or $w \neq 0$ at the end of Phase 1), no solution exists for (1) and you have the option of either terminating or continuing in so-called *Elastic mode* (see the discussion of the option **Elastic Mode** in Section 11.2). In elastic mode, a 'relaxed' or 'perturbed' problem is solved in which $f(x)$ is minimized while allowing some of the bounds to become 'elastic', i.e., to change from their specified values. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. You are able to assign which bounds will become elastic if elastic mode is ever started (see the argument **helast** in Section 5).

To make the relaxed problem meaningful, nag_opt_sparse_convex_qp_solve (e04nqc) minimizes $f(x)$ while (in some sense) finding the 'smallest' violation of the elastic variables. In the situation where all the variables are elastic, the relaxed problem has the form

Phase 2 (γ)

$$\underset{x,v,w}{\text{minimize}} f(x) + \gamma \sum_{j=1}^{n+m} (v_j + w_j) \quad \text{subject to } Ax - s = 0, \quad \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} - v + w \leq u, \quad v \geq 0, \quad w \geq 0,$$

where γ is a non-negative argument known as the elastic weight (see the option **Elastic Weight** in Section 11.2), and $f(x) + \gamma \sum_j (v_j + w_j)$ is called the *composite objective*. In the more general situation where only a subset of the bounds are elastic, the v's and w's for the non-elastic bounds are fixed at zero.

The *elastic weight* can be chosen to make the composite objective behave like either the original objective $f(x)$ or the sum of infeasibilities. If $\gamma = 0$, nag_opt_sparse_convex_qp_solve (e04nqc) will attempt to minimize f subject to the (true) upper and lower bounds on the non-elastic variables (and declare the problem infeasible if the non-elastic variables cannot be made feasible).

At the other extreme, choosing γ sufficiently large, will have the effect of minimizing the sum of the violations of the elastic variables subject to the original constraints on the non-elastic variables. Choosing a large value of the elastic weight is useful for defining a 'least-infeasible' point for an infeasible problem.

In Phase 1 and elastic mode, all calculations involving v and w are done implicitly in the sense that an elastic variable x_j is allowed to violate its lower bound (say) and an explicit value of v can be recovered as $v_j = l_j - x_j$.

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

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

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

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

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

Associated with each of the m equality constraints $Ax - s = 0$ is a *dual variable* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j (also known as a *reduced cost*). The reduced gradients for the variables x are the quantities $g - A^{\mathrm{T}}\pi$, where g is the gradient of the QP objective function; and the reduced gradients for the slack variables s are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds and $d_j = 0$ for all superbasic variables. In practice, an *approximate* QP solution is found by slightly relaxing these conditions on d_j (see the description of the option **Optimality Tolerance** in Section 11.2).

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

nag_opt_sparse_convex_qp_solve (e04nqc) is based on SQOPT, which is part of the SNOPT package described in Gill *et al.* (1999). It uses stable numerical methods throughout and includes a reliable basis package (for maintaining sparse LU factors of the basis matrix B), a practical anti-degeneracy procedure, efficient handling of linear constraints and bounds on the variables (by an active-set strategy), as well as automatic scaling of the constraints. Further details can be found in Section 10.

4 References

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

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

Gill P E, Murray W and Saunders M A (1995) User's guide for QPOPT 1.0: a Fortran package for quadratic programming *Report SOL 95-4* Department of Operations Research, Stanford University

Gill P E, Murray W and Saunders M A (1999) Users' guide for SQOPT 5.3: a Fortran package for large-scale linear and quadratic programming *Report SOL 99* Department of Operations Research, Stanford University

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

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

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

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

5 Arguments

The first n entries of the arguments **bl**, **bu**, **hs** and **x** refer to the variables x. The last m entries refer to the slacks s.

1: **start** – Nag_Start *Input*

On entry: indicates how a starting basis (and certain other items) are to be obtained.

start = **Nag_Cold** (Cold Start)

Requests that the Crash procedure be used to choose an initial basis, unless a basis file is provided via option **Old Basis File**, **Insert File** or **Load File** (see Section 11.2).

start = **Nag_BasisFile**

Is the same as **start** = **Nag_Cold** but is more meaningful when a basis file is given.

start = **Nag_Warm** (Warm Start)

Means that a basis is already defined in **hs** (probably from an earlier call).

Constraint: **start** = **Nag_BasisFile**, **Nag_Cold** or **Nag_Warm**.

2: **qphx** – function, supplied by the user *External Function*

For QP problems, you must supply a version of **qphx** to compute the matrix product Hx for the given vector x. If H has rows and columns 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. The number of columns of H_1 is specified in **ncolh**. For FP and LP problems, **qphx** will never be called by nag_opt_sparse_convex_qp_solve (e04nqc) and hence **qphx** may be specified as **NULL**.

Its specification is:

```
void qphx (Integer ncolh, const double x[], double hx[], Integer nstate,
     Nag_Comm *comm)
```

1: **ncolh** – Integer *Input*

On entry: this is the same argument **ncolh** as supplied to nag_opt_sparse_convex_qp_solve (e04nqc).

2: **x**[**ncolh**] – const double *Input*

On entry: the first **ncolh** elements of the vector x.

3: **hx**[**ncolh**] – double *Output*

On exit: the product Hx.

4: **nstate** – Integer *Input*

On entry: if **nstate** $= 1$, then nag_opt_sparse_convex_qp_solve (e04nqc) is calling **qphx** for the first time. This argument setting allows you to save computation time if certain data must be read or calculated only once. To preserve this data for a subsequent calculation place it in **comm** (below).

If **nstate** $= 0$ there is nothing special about the current call of **qphx**.

If **nstate** ≥ 2, then nag_opt_sparse_convex_qp_solve (e04nqc) is calling **qphx** for the last time. This argument setting allows you to perform some additional computation on the final solution. On the last call of **qphx**, if **nstate** $= 2$, the current x is optimal; if **nstate** $= 3$, the problem appears to be infeasible; if **nstate** $= 4$, the problem appears to be unbounded; and if **nstate** $= 5$, the iterations limit was reached.

5: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **qphx**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_sparse_convex_qp_solve (e04nqc) these pointers may be allocated memory by the user and initialized with various quantities for use by **qphx** when called from nag_opt_sparse_convex_qp_solve (e04nqc).

3: **m** – Integer *Input*

On entry: 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** below).

Constraint: **m** ≥ 1.

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 1.

5: **ne** – Integer *Input*

On entry: the number of non-zero elements in A.

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

6: **nname** – Integer *Input*

On entry: 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 $=$ **n** $+$ **m**

All names must be supplied.

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

7: **lenc** – Integer *Input*

On entry: the number of elements in the constant objective vector c.

Constraint: $0 \leq \textbf{lenc} \leq \textbf{n}$.

8: **ncolh** – Integer *Input*

On entry: n_H, the number of leading non-zero columns of the Hessian matrix H. For FP and LP problems, **ncolh** must be set to zero.

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

9: **iobj** – Integer *Input*

On entry: if $\textbf{iobj} > 0$, row **iobj** of A is a free row containing the non-zero elements of the vector c appearing in the linear objective term $c^{\mathrm{T}}x$.

If $\textbf{iobj} = 0$, there is no free row, i.e., the problem is either an FP problem, or a QP problem with $c = 0$.

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

10: **objadd** – double *Input*

On entry: the constant q, to be added to the objective for printing purposes. Typically $\textbf{objadd} = 0.0$.

11: **prob** – const char * *Input*

On entry: the name for the problem. It is used in the printed solution and in some functions that output basis files. Only the first eight characters of **prob** are significant.

12: **acol**[**ne**] – const double *Input*

On entry: the non-zero elements of A, ordered by increasing column index. Note that all elements must be assigned a value in the calling program.

13: **inda**[**ne**] – const Integer *Input*

On entry: $\textbf{inda}[i-1]$ must contain the row index of the non-zero element stored in $\textbf{acol}[i-1]$, for $i = 1, 2, \ldots, \textbf{ne}$. Thus a pair of values $(\textbf{acol}[k-1], \textbf{inda}[k-1])$ contains a matrix element and its corresponding row index.

If $\textbf{lenc} > 0$, the first **lenc** elements of **acol** and **inda** belong to variables corresponding to the constant objective term c.

If the problem has a quadratic objective, the first **ncolh** columns of **acol** and **inda** belong to variables corresponding to the non-zero block of the *QP* Hessian. Function **qphx** knows about these variables.

Note that the row indices for a column must lie in the range 1 to **m**, and may be supplied in any order.

Constraint: $1 \leq \textbf{inda}[i-1] \leq \textbf{m}$, for $i = 1, 2, \ldots, \textbf{ne}$.

14: **loca**[$\mathbf{n}+\mathbf{1}$] – const Integer *Input*

On entry: $\textbf{loca}[j-1]$ must contain the value $p+1$, where p is the index in **acol** and **inda** of the start of the jth column, for $j = 1, 2, \ldots, \textbf{n}$. Thus, the entries of column j are held in $\textbf{acol}[i]$, and their corresponding row indices are in $\textbf{inda}[i]$, for $i = k-1, k, \ldots, l-1$, where $k = \textbf{loca}[j-1]$ and $l = \textbf{loca}[j] - 1$. To specify the jth column as empty, set $\textbf{loca}[j-1] = \textbf{loca}[j]$. Note that the first and last elements of **loca** must be such that $\textbf{loca}[0] = 1$ and $\textbf{loca}[\textbf{n}] = \textbf{ne} + 1$. If your problem has no constraints, or just bounds on the variables, you may include a dummy 'free' row with a single (zero) element by setting $\textbf{acol}[0] = 0.0$, $\textbf{inda}[0] = 1$, $\textbf{loca}[0] = 1$, and $\textbf{loca}[j-1] = 2$, for $j = 1, 2 \ldots, \textbf{n}$. This row is made 'free' by setting its bounds to be $\textbf{bl}[n+1] = -bigbnd$ and $\textbf{bu}[n+1] = bigbnd$.

Constraints:

$\mathbf{loca}[0] = 1$;
$\mathbf{loca}[j] \geq 1$, for $j = 1, 2, \ldots, \mathbf{n} - 1$;
$\mathbf{loca}[\mathbf{n}] = \mathbf{ne} + 1$;
$0 \leq \mathbf{loca}[j + 1] - \mathbf{loca}[j] \leq \mathbf{m}$, for $j = 0, 1, \ldots, \mathbf{n} - 1$.

15: $\mathbf{bl}[\mathbf{n} + \mathbf{m}]$ – const double *Input*

On entry: 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 fix the jth variable, set $\mathbf{bl}[j-1] = \mathbf{bu}[j-1] = \beta$, say, where $|\beta| < bigbnd$. To specify a non-existent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}[j-1] \leq -bigbnd$, where $bigbnd$ is the value of the optional argument **Infinite Bound Size** (see Section 11.2). To specify the jth constraint as an *equality*, set $\mathbf{bl}[n+j-1] = \mathbf{bu}[n+j-1] = \beta$, say, where $|\beta| < bigbnd$. Note that the lower bound corresponding to the free row must be set to $-\infty$ and stored in $\mathbf{bl}[\mathbf{n} + \mathbf{iobj} - 1]$.

Constraint: if $\mathbf{iobj} > 0$, $\mathbf{bl}[\mathbf{n} + \mathbf{iobj} - 1] \leq -bigbnd$.

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

16: $\mathbf{bu}[\mathbf{n} + \mathbf{m}]$ – const double *Input*

On entry: 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, 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 non-existent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}[j-1] \geq bigbnd$. Note that the upper bound corresponding to the free row must be set to $+\infty$ and stored in $\mathbf{bu}[\mathbf{n} + \mathbf{iobj} - 1]$.

Constraints:

if $\mathbf{iobj} > 0$, $\mathbf{bu}[\mathbf{n} + \mathbf{iobj} - 1] \geq bigbnd$;
$\mathbf{bl}[i-1] \leq \mathbf{bu}[i-1]$ otherwise.

17: $\mathbf{c}[\mathbf{lenc}]$ – const double *Input*

On entry: contains the explicit objective vector c (if any). If the problem is of type FP, or if $\mathbf{lenc} = 0$, then **c** is not referenced and may be set to 0. (In that case, **c** may be dimensioned (1), or it could be any convenient array.)

18: $\mathbf{names}[\mathbf{nname}]$ – const char * *Input*

On entry: the optional column and row names, respectively.

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

If $\mathbf{nname} = \mathbf{n} + \mathbf{m}$, 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 $\mathbf{names}[\mathbf{n} + \mathbf{iobj} - 1]$.

Note: that only the first eight characters of the strings in **names** are significant.

19: $\mathbf{helast}[\mathbf{n} + \mathbf{m}]$ – const Integer *Input*

On entry: defines which variables are to be treated as being elastic in elastic mode. The allowed values of **helast** are:

$\mathbf{helast}[j-1]$	**Status in elastic mode**
0	Variable j is non-elastic and cannot be infeasible
1	Variable j can violate its lower bound
2	Variable j can violate its upper bound
3	Variable j can violate either its lower or upper bound

helast need not be assigned if optional argument **Elastic Mode** $= 0$ (see Section 11.2).

Constraint: **helast**$[j-1] = 0, 1, 2, 3$ if **Elastic Mode** $\neq 0$, for $j = 1, 2, \ldots,$ **n** $+$ **m**.

20: **hs**[**n** + **m**] – Integer *Input/Output*

On entry: if **start** = **Nag_Cold** or **Nag_BasisFile**, and a basis file of some sort is to be input (an **Old Basis File**, **Insert File** or **Load File**, see Section 11.2), then **hs** and **x** need not be set at all.

If **start** = **Nag_Cold** and there is no basis file, the first **n** elements of **hs** and **x** 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 $(A \quad -I)$. Possible values for **hs**$[j-1]$ are as follows:

hs$[j-1]$	**State of x**$[j-1]$ **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 **hs**$[j-1] = 0$ and **x**$[j-1] = 0.0$, 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 **hs**$[j-1] = 4$ and **x**$[j-1] =$ **bl**$[j-1]$ or **hs**$[j-1] = 5$ and **x**$[j-1] =$ **bu**$[j-1]$ as appropriate.

Following the Crash procedure, variables for which **hs**$[j-1] = 2$ are made superbasic. Other variables not selected for the basis are then made nonbasic at the value **x**$[j-1]$ if **bl**$[j-1] \leq$ **x**$[j-1] \leq$ **bu**$[j-1]$, or at the value **bl**$[j-1]$ or **bu**$[j-1]$ closest to **x**$[j-1]$.

If **start** = **Nag_Warm**, **hs** and **x** must specify the initial states and values, respectively, of the variables and slacks (x, s). If nag_opt_sparse_convex_qp_solve (e04nqc) has been called previously with the same values of **n** and **m**, **hs** already contains satisfactory information.

Constraints:

if **start** = **Nag_Cold**, $0 \leq$ **hs**$[j-1] \leq 5$, for $j = 1, 2, \ldots,$ **n**;
if **start** = **Nag_Warm**, $0 \leq$ **hs**$[j-1] \leq 3$, for $j = 1, 2, \ldots,$ **n** $+$ **m**.

On exit: the final states of the variables and slacks (x, s). The significance of each possible value of **hs**$[j-1]$ is as follows:

hs$[j-1]$	**State of variable** j	**Normal value of x**$[j-1]$
0	Nonbasic	**bl**$[j-1]$
1	Nonbasic	**bu**$[j-1]$
2	Superbasic	Between **bl**$[j-1]$ and **bu**$[j-1]$
3	Basic	Between **bl**$[j-1]$ and **bu**$[j-1]$

If **ninf** $= 0$, basic and superbasic variables may be outside their bounds by as much as the value of the optional argument **Feasibility Tolerance** (see Section 11.2). Note that unless the optional argument **Scale Option** $= 0$ (see Section 11.2) 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 **x**$[j-1]$ lies strictly between its bounds.

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

21: **x**[**n** + **m**] – double *Input/Output*

On entry: the initial values of the variables x, if **start** = **Nag_Warm** and slacks s, i.e., (x, s). (See the description for **hs** above.)

On exit: the final values of the variables and slacks (x, s).

22: **pi**[**m**] – double *Output*

On exit: contains the dual variables π (a set of Lagrange-multipliers (shadow prices) for the general constraints).

23: **rc**[**n** + **m**] – double *Output*

On exit: the first **n** elements contain the reduced costs, $g - \begin{pmatrix} A & -I \end{pmatrix}^{\mathrm{T}} \pi$, where g is the gradient of the objective if **x** is feasible (or the gradient of the Phase 1 objective otherwise). The last **m** entries are π.

24: **ns** – Integer * *Input/Output*

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

On exit: the final number of superbasics. This will be zero for FP and LP problems.

25: **ninf** – Integer * *Output*

On exit: the number of infeasibilities.

26: **sinf** – double * *Output*

On exit: the sum of the scaled infeasibilities. This will be zero if **ninf** = 0, and is most meaningful when **Scale Option** = 0 (see Section 11.2).

27: **obj** – double * *Output*

On exit: 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.

28: **state** – Nag_E04State * *Communication Structure*

Note: **state** is a NAG defined type (see Section 2.2.1.1 of the Essential Introduction).

state contains internal information required for functions in this suite. It must not be modified in any way.

29: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 2.2.1.1 of the Essential Introduction).

30: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Internal memory allocation failed when attempting to obtain workspace sizes ⟨*value*⟩, ⟨*value*⟩ and ⟨*value*⟩. Please contact NAG.

NE_ALLOC_INSUFFICIENT

Internal memory allocation was insufficient. Please contact NAG.

NE_ARRAY_INPUT

On entry, **loca**[0] is not 1 or **loca**[$\langle value \rangle$] is not equal to **ne** $+ 1$. **loca**[0] $= \langle value \rangle$, **loca**[$\langle value \rangle$] $= \langle value \rangle$, **ne** $= \langle value \rangle$.

On entry, row index $\langle value \rangle$ in **inda**[$\langle value \rangle$] is outside the range 1 to **m** $= \langle value \rangle$.

NE_BAD_PARAM

Basis file dimensions do not match this problem.

NE_BASIS_FAILURE

An error has occurred in the basis package, perhaps indicating incorrect setup of arrays **inda** and **loca**. Set the optional argument **Print File** and examine the output carefully for further information.

NE_BASIS_ILL_COND

Numerical difficulties have been encountered and no further progress can be made.

NE_BASIS_SINGULAR

The basis is singular after several attempts to factorize it (and add slacks where necessary).

NE_E04NPC_NOT_INIT

Initialization function nag_opt_sparse_convex_qp_init (e04npc) has not been called.

NE_HESS_INDEF

Error in the user-supplied function **qphx**: the QP Hessian is indefinite.

NE_HESS_TOO_BIG

The superbasics limit is too small.

NE_INT

On entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 1.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 1.

NE_INT_2

On entry, **iobj** < 0 or **iobj** $>$ **m**. **iobj** $= \langle value \rangle$, **m** $= \langle value \rangle$.

On entry, **lenc** < 0 or **lenc** $>$ **n**. **lenc** $= \langle value \rangle$, **n** $= \langle value \rangle$.

On entry, **ncolh** < 0 or **ncolh** $>$ **n**. **ncolh** $= \langle value \rangle$, **n** $= \langle value \rangle$.

On entry, **ne** is not equal to the number of non-zeros in **acol**. **ne** $= \langle value \rangle$, non-zeros in **acol** $= \langle value \rangle$.

NE_INT_3

On entry, **n** $= \langle value \rangle$, **m** $= \langle value \rangle$, **nname** $= \langle value \rangle$.
Constraint: **nname** $= 1$ or **n** $+$ **m**.

On entry, **ne** < 1 or **ne** $>$ **n** $\times$ **m**. **ne** $= \langle value \rangle$, **n** $= \langle value \rangle$, **m** $= \langle value \rangle$.

On entry, **nname** is not equal to 1 or **n** $+$ **m**. **nname** $= \langle value \rangle$, **n** $= \langle value \rangle$, **m** $= \langle value \rangle$.

NE_INTERNAL_ERROR

An unexpected error has occurred. Set the optional argument **Print File** and examine the output carefully for further information.

NE_NOT_REQUIRED_ACC

The requested accuracy could not be achieved.

NE_REAL_2

On entry, bounds **bl** and **bu** for $\langle value \rangle$ are equal and infinite. **bl** $=$ **bu** $= \langle value \rangle$, $bigbnd = \langle value \rangle$.

On entry, bounds **bl** and **bu** for $\langle value \rangle\langle value \rangle$ are equal and infinite. **bl** $=$ **bu** $= \langle value \rangle$, $bigbnd = \langle value \rangle$.

On entry, bounds for $\langle value \rangle$ are inconsistent. **bl** $= \langle value \rangle$, **bu** $= \langle value \rangle$.

On entry, bounds for $\langle value \rangle\langle value \rangle$ are inconsistent. **bl** $= \langle value \rangle$, **bu** $= \langle value \rangle$.

NE_UNBOUNDED

The problem appears to be unbounded. The constraint violation limit has been reached.

The problem appears to be unbounded. The objective function is unbounded.

NW_NOT_FEASIBLE

The linear constraints appear to be infeasible.

The problem appears to be infeasible. Infeasibilites have been minimized.

The problem appears to be infeasible. Nonlinear infeasibilites have been minimized.

The problem appears to be infeasible. The linear equality constraints could not be satisfied.

NW_SOLN_NOT_UNIQUE

Weak solution found – the solution is not unique.

NW_TOO_MANY_ITER

Iteration limit reached.

Major iteration limit reached.

7 Accuracy

nag_opt_sparse_convex_qp_solve (e04nqc) 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

If **Print Level** > 0, one line of information is output to the **Print File** every kth iteration, where k is the specified **Print Frequency** (see Section 11.2). A heading is printed before the first such line following a basis factorization. The heading contains the items described below. In this description, a pricing operation is defined to be the process by which one or more nonbasic variables are selected to become superbasic (in addition to those already in the superbasic set). The variable selected will be denoted by `jq`. If the problem is purely linear, variable `jq` will usually become basic immediately (unless it should happen to reach its opposite bound and return to the nonbasic set).

If **Partial Price** (see Section 11.2) is in effect, variable `jq` is selected from A_{pp} or I_{pp}, the ppth segments of the constraint matrix $(A \quad -I)$.

Label	Description
`Itn`	is the iteration count.
`pp`	is the optional 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. Algebraically, `dj` is $d_j = g_j - \pi^{\mathrm{T}} a_j$, for $j = \mathtt{jq}$ where g_j is the gradient of the current objective function, π is the vector of dual variables, and a_j is the jth column of the constraint matrix $(A \quad -I)$. Note that `dj` is the norm of the reduced-gradient vector at the start of the iteration, just after the pricing operation.
`+SBS`	is the variable `jq` 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 non-zero, otherwise it becomes nonbasic.
`-BS`	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 `-SBS` has become basic if this entry is non-zero, 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 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 $(A \quad -I)$ replaces the rth column of the basis matrix B. Wherever possible, `Step` is chosen so as to avoid extremely small values of `Pivot` (since they may cause the basis to be nearly singular). In extreme cases, it may be necessary to increase the value of the optional argument **Pivot Tolerance** (see Section 11.2) to exclude very small elements of y from consideration during the computation of `Step`.
`Ninf`	is the number of violated constraints (infeasibilities). This will be zero during the optimality phase.
`Sinf/Objective`	is the value of the current objective function. If x is not feasible, `Sinf` gives the sum of the magnitudes of constraint violations. If x is feasible, `Objective` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point. During the optimality phase, the value of the objective function will be non-increasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists.
`L`	is the number of non-zeros in the basis factor L. Immediately after a basis factorization $B = LU$, this entry contains `lenL` (see Section 12). Further non-zeros are added to `L` when various columns of B are later replaced. (Thus, `L` increases monotonically.)

`U`	is the number of non-zeros in the basis factor U. Immediately after a basis factorization $B = LU$, this entry contains `lenU` (see Section 12). 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.

The following will be output if the problem is QP or if the superbasic is non-empty (i.e., if the current solution is nonbasic).

Label	**Description**
`Norm rg`	is $\|d_S\|$, the Euclidean norm of the reduced gradient (see Section 10.3). During the optimality phase, this norm will be approximately zero after a unit step.
`Ns`	is the current number of superbasic variables.
`Cond Hz`	is a lower bound on the condition number of the reduced Hessian (see Section 10.2). The larger this number, the more difficult the problem. Attention should be given to the scaling of the variables and the constraints to guard against high values of `Cond Hz`.

9 Example

To minimize 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

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

and to the linear constraints

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

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

$$x^* = (0.0, 349.40, 648.85, 172.85, 407.52, 271.36, 150.02)^{\mathrm{T}}.$$

One bound constraint and four linear constraints are active at the solution. Note that the Hessian matrix H is positive semi-definite.

9.1 Program Text

```
/* nag_nag_opt_sparse_convex_qp_solve (e04nqc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 8, 2004.
 */

#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

static void qphx(Integer ncolh, const double x[], double hx[],
                 Integer nstate, Nag_Comm *comm);

int main(void)
{

  /* Scalars */
  double obj, objadd, sinf;
  Integer exit_status, i, icol, iobj, j, jcol, lenc, m, n, ncolh, ne, ninf;
  Integer nname, ns;

  /* Arrays */
  char prob[9], start_char[2];
  char **names;
  double *acol=0, *bl=0, *bu=0, *c=0, *pi=0, *rc=0, *x=0;
  Integer *helast=0, *hs=0, *inda=0, *loca=0;

  /*Nag Types*/
  Nag_E04State state;
  NagError fail;
  Nag_Start start;
  Nag_Comm comm;
  Nag_FileID fileid;

  exit_status = 0;
  INIT_FAIL(fail);

  Vprintf("nag_opt_sparse_convex_qp_solve (e04nqc) Example Program Results\n");

  /* Skip heading in data file. */
  Vscanf("%*[^\n] ");

  /* Read ne, iobj, ncolh, start and nname from data file. */
  Vscanf("%ld%ld%*[^\n] ",   &n,  &m);
  Vscanf("%ld%ld%ld ' %1s '%ld%*[^\n] ",
         &ne,  &iobj,  &ncolh,  start_char,  &nname);
  if (n>=1 && m >= 1)
    {
      /* Allocate memory */
      if ( !(names = NAG_ALLOC(n+m, char *)) ||
           !(acol = NAG_ALLOC(ne, double)) ||
           !(bl = NAG_ALLOC(m+n, double)) ||
           !(bu = NAG_ALLOC(m+n, double)) ||
           !(c = NAG_ALLOC(1, double)) ||
           !(pi = NAG_ALLOC(m, double)) ||
           !(rc = NAG_ALLOC(n+m, double)) ||
           !(x = NAG_ALLOC(n+m, double)) ||
           !(helast = NAG_ALLOC(n+m, Integer)) ||
           !(hs = NAG_ALLOC(n+m, Integer)) ||
           !(inda = NAG_ALLOC(ne, Integer)) ||
           !(loca = NAG_ALLOC(n+1, Integer)) )
```

```
      {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
  }
else
  {
    Vprintf("%s", "Either m or n invalid\n");
    exit_status = 1;
    return exit_status;
  }

/* Read names from data file. */
for (i = 1; i <= nname; ++i)
  {
    names[i-1] = NAG_ALLOC(9, char);
    Vscanf(" ' %8s '", names[i-1]);
  }
Vscanf("%*[^\n] ");

/* Read the matrix acol from data file. Set up LOCA. */
jcol = 1;
loca[jcol - 1] = 1;
for (i = 1; i <= ne; ++i)
  {
    /* Element (inda[i-1], icol) is stored in acol[i-1]. */
    Vscanf("%lf%ld%ld%*[^\n] ", &acol[i - 1], &inda[i - 1],
           &icol);
    if (icol < jcol)
      {
        /* Elements not ordered by increasing column index. */
        Vprintf("%s%5ld%s%5ld%s%s\n", "Element in column",
                icol, " found after element in column", jcol, ". Problem",
                " abandoned.");
      }
    else if (icol == jcol + 1)
      {
        /* Index in ACOL of the start of the ICOL-th column equals I. */
        loca[icol - 1] = i;
        jcol = icol;
      }
    else if (icol > jcol + 1)
      {
        /* Index in acol of the start of the icol-th column equals i, */
        /* but columns jcol+1,jcol+2,...,icol-1 are empty. Set the */
        /* corresponding elements of loca to i. */
        for (j = jcol + 1; j <= icol - 1; ++j)
          {
            loca[j - 1] = i;
          }
        loca[icol - 1] = i;
        jcol = icol;
      }
  }
loca[n] = ne + 1;

if (n > icol)
  {
    /* Columns n,n-1,...,icol+1 are empty. Set the corresponding */
    /* elements of loca accordingly. */
    for (i = n; i >= icol + 1; --i)
      {
        loca[i - 1] = loca[i];
      }
  }

/* Read bl, bu, hs and x from data file. */
for (i = 1; i <= n + m; ++i)
  {
    Vscanf("%lf", &bl[i - 1]);
```

```
    }
  Vscanf("%*[^\n] ");

  for (i = 1; i <= n + m; ++i)
    {
      Vscanf("%lf", &bu[i - 1]);
    }
  Vscanf("%*[^\n] ");

  if (*(unsigned char *)start_char == 'C')
    {
      start = Nag_Cold;
      for (i = 1; i <= n; ++i)
        {
          Vscanf("%ld", &hs[i - 1]);
        }
      Vscanf("%*[^\n] ");
    }
  else if (*(unsigned char *)start_char == 'W')
    {
      start = Nag_Warm;
      for (i = 1; i <= n + m; ++i)
        {
          Vscanf("%ld", &hs[i - 1]);
        }
      Vscanf("%*[^\n] ");
    }
  for (i = 1; i <= n; ++i)
    {
      Vscanf("%lf", &x[i - 1]);
    }
  Vscanf("%*[^\n] ");

  /* Call nag_opt_sparse_convex_qp_init (e04npc) to initialise e04nqc. */
  /* nag_opt_sparse_convex_qp_init (e04npc).
   * Initialization function for
   * nag_opt_sparse_convex_qp_solve (e04nqc)
   */
  nag_opt_sparse_convex_qp_init(&state, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Initialisation of nag_opt_sparse_convex_qp_solve (e04nqc)"
              " failed.\n");
      exit_status = 1;
      goto END;
    }
  /* By default nag_opt_sparse_convex_qp_solve (e04nqc) does not print
   *  monitoring information. Call nag_open_file (x04acc) to set the print file
   * fileid */
  /* nag_open_file (x04acc).
   * Open unit number for reading, writing or appending, and
   * associate unit with named file
   */
  nag_open_file("", 2, &fileid, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Fileid could not be obtained.\n");
      exit_status = 1;
      goto END;
    }

  /* nag_opt_sparse_convex_qp_option_set_integer (e04ntc).
   * Set a single option for nag_opt_sparse_convex_qp_solve
   * (e04nqc) from an integer argument
   */
  nag_opt_sparse_convex_qp_option_set_integer("Print file", fileid, &state,
                                              &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Files stream could not be set.\n");
      exit_status = 1;
```

```
      goto END;
    }

  /* We have no explicit objective vector so set lenc = 0; the
   * objective vector is stored in row iobj of acol.
   */
  lenc = 0;
  objadd = 0.;
  strcpy(prob, "");

  /* Do not allow any elastic variables (i.e. they cannot be */
  /* infeasible). If we'd set optional argument "Elastic mode" to 0, */
  /* we wouldn't need to set the individual elements of array helast. */
  for (i = 1; i <= n + m; ++i)
    {
      helast[i - 1] = 0;
    }

  /* Solve the QP problem. */
  /* nag_opt_sparse_convex_qp_solve (e04nqc).
   * LP or QP problem (suitable for sparse problems)
   */
  nag_opt_sparse_convex_qp_solve(start, qphx, m, n, ne, nname, lenc, ncolh,
                                 iobj, objadd, prob, acol, inda, loca, bl, bu,
                                 c, names, helast,  hs, x, pi, rc, &ns, &ninf,
                                 &sinf, &obj, &state, &comm, &fail);

  Vprintf("\n");
  Vprintf("On exit from e04nqc, fail.message = %s\n", fail.message);
  if (fail.code == NE_NOERROR)
    {
      Vprintf("Final objective value = %11.3e\n",   obj);
      Vprintf("Optimal X = ");

      for (i = 1; i <= n; ++i)
        {
          Vprintf("%9.2f%s", x[i - 1], i%7 == 0 || i == n ?"\n":" ");
        }
    }
 END:
  for (i = 0; i < n+m; i++)
    {
      if (names[i]) NAG_FREE(names[i]);
    }
  if (names) NAG_FREE(names);
  if (acol) NAG_FREE(acol);
  if (bl) NAG_FREE(bl);
  if (bu) NAG_FREE(bu);
  if (c) NAG_FREE(c);
  if (pi) NAG_FREE(pi);
  if (rc) NAG_FREE(rc);
  if (x) NAG_FREE(x);
  if (helast) NAG_FREE(helast);
  if (hs) NAG_FREE(hs);
  if (inda) NAG_FREE(inda);
  if (loca) NAG_FREE(loca);
  return exit_status;
}

static void qphx(Integer ncolh, const double x[], double hx[],
                 Integer nstate, Nag_Comm *comm)
{
  /* Routine to compute H*x. (In this version of qphx, the Hessian
   * matrix H is not referenced explicitly.)
   */

  /* Parameter adjustments */
#define HX(I) hx[(I)-1]
#define X(I) x[(I)-1]

  /* Function Body */
```

```
  HX(1) = X(1) * 2;
  HX(2) = X(2) * 2;
  HX(3) = (X(3) + X(4)) * 2;
  HX(4) = HX(3);
  HX(5) = X(5) * 2;
  HX(6) = (X(6) + X(7)) * 2;
  HX(7) = HX(6);
  return;
} /* qphx */
```

9.2 Program Data

```
nag_opt_sparse_convex_qp_solve (e04nqc) Example Program Data
 7  8                          : Values of N and M
48  8  7  'C'  15              : Values of NNZ, IOBJ, NCOLH, START and NNAME

'...X1...'  '...X2...'  '...X3...'  '...X4...'  '...X5...'
'...X6...'  '...X7...'  '..ROW1..'  '..ROW2..'  '..ROW3..'
'..ROW4..'  '..ROW5..'  '..ROW6..'  '..ROW7..'  '..COST..' : End of array NAMES

    0.02   7   1   : Sparse matrix A, ordered by increasing column index;
    0.02   5   1   : each row contains ACOL(i), INDA(i), ICOL (= column index)
    0.03   3   1   : The row indices may be in any order. In this example
    1.00   1   1   : row 8 defines the linear objective term transpose(C)*X.
    0.70   6   1
    0.02   4   1
    0.15   2   1
 -200.00   8   1
    0.06   7   2
    0.75   6   2
    0.03   5   2
    0.04   4   2
    0.05   3   2
    0.04   2   2
    1.00   1   2
-2000.00   8   2
    0.02   2   3
    1.00   1   3
    0.01   4   3
    0.08   3   3
    0.08   7   3
    0.80   6   3
-2000.00   8   3
    1.00   1   4
    0.12   7   4
    0.02   3   4
    0.02   4   4
    0.75   6   4
    0.04   2   4
-2000.00   8   4
    0.01   5   5
    0.80   6   5
    0.02   7   5
    1.00   1   5
    0.02   2   5
    0.06   3   5
    0.02   4   5
-2000.00   8   5
    1.00   1   6
    0.01   2   6
    0.01   3   6
    0.97   6   6
    0.01   7   6
  400.00   8   6
    0.97   7   7
    0.03   2   7
    1.00   1   7
  400.00   8   7        : End of matrix A

 0.0       0.0       4.0E+02   1.0E+02   0.0       0.0
```

```
0.0       2.0E+03  -1.0E+25  -1.0E+25  -1.0E+25  -1.0E+25
1.5E+03   2.5E+02  -1.0E+25             : End of lower bounds array BL

2.0E+02   2.5E+03   8.0E+02   7.0E+02  1.5E+03  1.0E+25
1.0E+25   2.0E+03   6.0E+01   1.0E+02  4.0E+01  3.0E+01
1.0E+25   3.0E+02   1.0E+25             : End of upper bounds array BU

0    0    0    0    0    0    0         : Initial array HS
0.0  0.0  0.0  0.0  0.0  0.0  0.0       : Initial vector X
```

9.3 Program Results

```
nag_opt_sparse_convex_qp_solve (e04nqc) Example Program Results

 Parameters
 ==========

 Files
 -----
 Solution file..........        0       Old basis file ........        0       (Print file)...........        6
 Insert file............        0       New basis file ........        0       (Summary file).........        0
 Punch file.............        0       Backup basis file......        0
 Load file..............        0       Dump file..............        0

 Frequencies
 -----------
 Print frequency........      100       Check frequency........       60       Save new basis map.....      100
 Summary frequency......      100       Factorization frequency       50       Expand frequency.......    10000

 LP/QP Parameters
 ----------------
 Minimize...............                QPsolver Cholesky......                Cold start.............
 Scale tolerance........    0.900       Feasibility tolerance..  1.00E-06      Iteration limit........    10000
 Scale option...........        2       Optimality tolerance...  1.00E-06      Print level............        1
 Crash tolerance........    0.100       Pivot tolerance........  2.05E-11      Partial price..........        1
 Crash option...........        3       Elastic weight.........  1.00E+00      Prtl price section ( A)        7
 Elastic mode...........        1       Elastic objective......        1       Prtl price section (-I)        8

 QP objective
 ------------
 Objective variables....        7       Hessian columns........        7       Superbasics limit......        7
 Nonlin Objective vars..        7       Unbounded step size....  1.00E+20
 Linear Objective vars..        0

 Miscellaneous
 -------------
 LU factor tolerance....     3.99       LU singularity tol.....  2.05E-11      Timing level...........        0
 LU update tolerance....     3.99       LU swap tolerance......  1.03E-04      Debug level............        0
 LU partial  pivoting...                eps (machine precision)  1.11E-16      System information.....       No

 Nonlinear constraints       0     Linear constraints       8
 Nonlinear variables         7     Linear variables         0
 Jacobian  variables         0     Objective variables      7
 Total constraints           8     Total variables          7


 Itn      1: Feasible linear constraints

 E04NQF EXIT   0 -- finished successfully
 E04NQF INFO   1 -- optimality conditions satisfied

 Problem name
 No. of iterations                   9   Objective value     -1.8477846771E+06
 No. of Hessian products            16   Objective row       -2.9886903537E+06
                                         Quadratic objective  1.1409056766E+06
 No. of superbasics                  2   No. of basic nonlinears           4
 No. of degenerate steps             0   Percentage                     0.00
 Max x       (scaled)        3 2.4E-01   Max pi      (scaled)        6 4.7E+07
 Max x                       3 6.5E+02   Max pi                      7 1.5E+04
 Max Prim inf(scaled)        0 0.0E+00   Max Dual inf(scaled)        6 1.1E-08
```

```
 Max Primal infeas          0 0.0E+00   Max Dual infeas            9 6.4E-12

 Name                                   Objective Value     -1.8477846771E+06

 Status         Optimal Soln            Iteration      9    Superbasics     2

 Section 1 - Rows

  Number  ...Row.. State  ...Activity...  Slack Activity  ..Lower Limit.  ..Upper Limit.  .Dual Activity    ..i

       8  ..ROW1..    EQ      2000.00000           .          2000.00000      2000.00000    -12900.76766      1
       9  ..ROW2..    BS        49.23160       -10.76840           None         60.00000         0.00000      2
      10  ..ROW3..    UL       100.00000           .               None        100.00000     -2324.86620      3
      11  ..ROW4..    BS        32.07187        -7.92813           None         40.00000          .           4
      12  ..ROW5..    BS        14.55719       -15.44281           None         30.00000          .           5
      13  ..ROW6..    LL      1500.00000           .          1500.00000           None      14454.60290      6
      14  ..ROW7..    LL       250.00000           .           250.00000       300.00000     14580.95432      7
      15  ..COST..    BS -2988690.35370  -2988690.35370           None            None           -1.0         8

 Section 2 - Columns

  Number  .Column. State  ...Activity...  .Obj Gradient.  ..Lower Limit.  ..Upper Limit.  Reduced Gradnt    m+j

       1  ...X1...    LL          .           -200.00000           .          200.00000      2360.67253      9
       2  ...X2...    BS       349.39923     -1301.20153           .         2500.00000        -0.00000     10
       3  ...X3...   SBS       648.85342      -356.59829       400.00000       800.00000         0.00000     11
       4  ...X4...   SBS       172.84743      -356.59829       100.00000       700.00000        -0.00000     12
       5  ...X5...    BS       407.52089     -1184.95822           .         1500.00000        -0.00000     13
       6  ...X6...    BS       271.35624      1242.75804           .              None          -0.00000     14
       7  ...X7...    BS       150.02278      1242.75804           .              None           0.00000     15

On exit from e04nqc, fail.message = NE_NOERROR:
  No error.
Final objective value =  -1.848e+06
Optimal X =      0.00    349.40    648.85    172.85    407.52    271.36    150.02
```

Note: *the remainder of this document is intended for more advanced users. Section 10 contains a detailed algorithm description that may be needed in order to understand Sections 11 and 12. Section 11 describes the optional arguments that may be set by calls to nag_opt_sparse_convex_qp_option_set_file (e04nrc), nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) and/or nag_opt_sparse_convex_qp_option_set_double (e04nuc). Section 12 describes the quantities that can be requested to monitor the course of the computation.*

10 Algorithmic Details

This section contains a description of the method used by nag_opt_sparse_convex_qp_solve (e04nqc).

10.1 Overview

nag_opt_sparse_convex_qp_solve (e04nqc) is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method is similar to that of Gill and Murray (1978), and is described in detail by Gill *et al.* (1991). Here we briefly summarize the main features of the method. Where possible, explicit reference is made to the names of variables that are 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 12) to the quadratic objective function (the printed quantity `Objective`; see Section 12).

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

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

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

10.2 Definition of the Working Set and Search Direction

At each iterate (x, s), a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the value of the optional argument **Feasibility Tolerance**; see Section 11.2). The working set is the current prediction of the constraints that hold with equality at a solution of the LP or QP problem. Let m_W denote the number of constraints in the working set (including bounds), and let W denote the associated m_W by $(n + m)$ *working set matrix* consisting of the m_W gradients of the working set constraints.

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

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

This characterization allows p to be computed using any n by n_Z full-rank matrix Z that spans the null space of W. (Thus, $n_Z = n - m_W$ and $WZ = 0$.) The null space matrix Z is defined from a sparse LU factorization of part of W (see (6) and (7) 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 $(A \quad -I)$ and the unit rows associated with the upper and lower bounds in the working set.

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

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

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

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

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

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

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

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

This matrix is used only as an operator, i.e., it is never computed explicitly. Products of the form Zv and

$Z^T g$ 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:

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

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

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

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

If the initial R is singular, enough variables are fixed at their current value to give a non-singular 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 non-singular).

10.3 Main Iteration

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

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

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 argument **Optimality Tolerance** (see Section 11.2). If a multiplier is non-optimal, 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 non-zero, there is no feasible point and the function terminates immediately with **fail.code** = **NE_NOT_REQUIRED_ACC** (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

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

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

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

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

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

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 α_N, the step to the 'nearest' constraint along p. This constraint is then added to the working set at the next iteration.

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

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

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 **fail.code** = **NE_UNBOUNDED** (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.

nag_opt_sparse_convex_qp_solve (e04nqc) makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

$$\underset{x,v,w}{\text{minimize}}\, e^{\mathrm{T}}(v + w) \quad \text{subject to } l \leq \left\{ \begin{array}{c} x \\ Gx - v + w \end{array} \right\} \leq u, \quad v \geq 0, \quad w \geq 0, \tag{13}$$

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*.)

10.4 Miscellaneous

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

$$(B \quad S)^{\mathrm{T}} = LU, \tag{14}$$

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

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

Suppose that the value of the optional argument **Feasibility Tolerance** (see Section 11.2) is δ. Over a period of K iterations (where K is the value of the optional argument **Expand Frequency**; see Section 11.2), the feasibility tolerance actually used by the function (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/K$).

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

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

The resetting procedure is also invoked when the function reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any non-trivial 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 $\alpha \leq \alpha_N$) 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 Arguments

Several optional arguments in nag_opt_sparse_convex_qp_solve (e04nqc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_opt_sparse_convex_qp_solve (e04nqc) these optional arguments have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional arguments 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 arguments. A complete list of optional arguments and their default values is given in Section 11.1.

Optional arguments may be specified by calling one, or any, of the functions nag_opt_sparse_convex_qp_option_set_file (e04nrc), nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) and nag_opt_sparse_convex_qp_option_set_double (e04nuc) prior to a call to nag_opt_sparse_convex_qp_solve (e04nqc), but after a call to nag_opt_sparse_convex_qp_init (e04npc).

nag_opt_sparse_convex_qp_option_set_file (e04nrc) reads options from an external options file, with `Begin` and `End` as the first and last lines respectively and each intermediate line defining a single optional argument. For example,

```
Begin
   Print Level = 5
End
```

The call

```
e04nrc (ioptns, &state, &fail);
```

can then be used to read the file on descriptor `ioptns`. **fail.code** = **NE_NOERROR** on successful exit. nag_opt_sparse_convex_qp_option_set_file (e04nrc) should be consulted for a full description of this method of supplying optional arguments.

nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) or nag_opt_sparse_convex_qp_option_set_double (e04nuc) can be called to supply options directly, one call being necessary for each optional argument. nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) or nag_opt_sparse_convex_qp_option_set_double (e04nuc) should be consulted for a full description of this method of supplying optional arguments.

All optional arguments not specified by you are set to their default values. Optional arguments specified by you are unaltered by nag_opt_sparse_convex_qp_solve (e04nqc) (unless they define invalid values) and so remain in effect for subsequent calls unless altered by you.

11.1 Optional Argument Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 11.2. The minimum abbreviation of each keyword is underlined. The qualifier may be omitted. The letters i and r denote Integer and double values required with certain options. The default value of an option is used whenever the condition $|i| \geq 100000000$ is satisfied. The number ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

Optional arguments used to specify files (e.g., **Dump File** and **Print File**) have type **Nag_FileID**. This ID value must either be set to 0 (the default value) in which case there will be no output, or will be as returned by a call of nag_open_file (x04acc).

Optional Arguments	Default Values
Backup Basis File	Default $= 0$
Check Frequency	Default $= 60$
Crash Option	Default $= 3$
Crash Tolerance	Default $= 0.1$
Defaults	
Dump File	Default $= 0$
Elastic Mode	Default $= 1$
Elastic Objective	Default $= 1$
Elastic Weight	Default $= 1.0$
Expand Frequency	Default $= 10000$
Factorization Frequency	Default $= 100(\text{LP})$ or $50(\text{QP})$
Feasibility Tolerance	Default $= 10^{-6}$
Infinite Bound Size	Default $= 10^{20}$
Insert File	Default $= 0$
Iteration Limit	Default $= \max(10000, m)$
Iters	
Itns	
List	Default $=$ **Nolist**
Load File	Default $= 0$
LU Factor Tolerance	Default $= 3.99$
LU Singularity Tolerance	Default $= \epsilon^{0.67}$
LU Update Tolerance	Default $= 3.99$
Maximize	Default $=$ **Minimize**
Minimize	
New Basis File	Default $= 0$
Nolist	
Optimality Tolerance	Default $= 10^{-6}$
Old Basis File	Default $= 0$
Partial Price	Default $= 10(\text{LP})$ or $1(\text{QP})$
Pivot Tolerance	Default $= \epsilon^{0.67}$
Print File	Default $= 0$
Print Frequency	Default $= 100$
Print Level	Default $= 1$
Punch File	Default $= 0$
Save Frequency	Default $= 100$
Scale Option	Default $= 2$
Scale Tolerance	Default $= 0.9$
Solution File	Default $= 0$
Summary File	Default $= 0$
Summary Frequency	Default $= 100$
Superbasics Limit	Default $= \min(500, n_H + 1, n)$
Suppress Parameters	
Timing Level	Default $= 0$
Unbounded Step Size	Default $= \max(bigbnd, 10^{20})$

11.2 Description of the Optional Arguments

Check Frequency – Integer 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.

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

Crash Option – Integer i Default $= 3$
Crash Tolerance – double r Default $= 0.1$

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

If **start** $=$ **Nag_Cold**, 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.

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 the matrix A.
2	The Crash procedure is called once, looking for a triangular basis in rows.
3	The Crash procedure is called twice. The two calls treat linear equalities and linear inequalities separately.

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.

This value allows the Crash procedure to ignore certain 'small' non-zero elements in each column of A. If $a_{\max}$ is the largest element in column j, other non-zeros a_{ij} in the column are ignored if $\left|a_{ij}\right| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, 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 sooner on some problems.

For example, suppose the first m columns of A form the matrix shown under LU factor tolerance; i.e., a tridiagonal matrix with entries -1, 4, -1. To help the Crash procedure choose all m columns for the initial basis, we would specify Crash tolerance r for some value of $r > \frac{1}{4}$.

Defaults

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

Dump File – Nag_FileID i_1 Default $= 0$
Load File – Nag_FileID i_2 Default $= 0$

(See Section 11.1 for a description of **Nag_FileID**.)

Dump File and **Load File** are similar to **Punch File** and **Insert File**, but they record solution information in a manner that is more direct and more easily modified. A full description of information recorded in **Dump File** and **Load File** is given in Gill *et al.* (1999).

If **Dump File** > 0, the last solution obtained will be output to the file **Dump File**.

If **Load File** > 0, the **Load File** containing basis information will be read. The file will usually have been output previously as a **Dump File**. The file will not be accessed if an **Old Basis File** or an **Insert File** is specified.

Elastic Mode – Integer i Default $= 1$

This argument determines if (and when) elastic mode is to be started. Three elastic modes are available as follows:

i	**Meaning**
0	Elastic mode is never invoked. nag_opt_sparse_convex_qp_solve (e04nqc) will terminate as soon as infeasibility is detected. There may be other points with significantly smaller sums of infeasibilities.
1	Elastic mode is invoked only if the constraints are found to be infeasible (the default). If the constraints are infeasible, continue in elastic mode with the composite objective determined by the values of **Elastic Objective** and **Elastic Weight**.
2	The iterations start and remain in elastic mode. This option allows you to minimize the composite objective function directly without first performing Phase 1 iterations.

The success of this option will depend critically on your choice of **Elastic Weight**. If **Elastic Weight** is sufficiently large and the constraints are feasible, the minimizer of the composite objective and the solution of the original problem are identical. However, if the **Elastic Weight** is not sufficiently large, the minimizer of the composite function may be infeasible, even though a feasible point for the constraints may exist.

Elastic Objective – Integer i Default $= 1$

This option determines the form of the composite objective. Three types of composite objectives are available.

i	**Meaning**
0	Include only the true objective $f(x)$ in the composite objective. This option sets $\gamma = 0$ in the composite objective and will allow nag_opt_sparse_convex_qp_solve (e04nqc) to ignore the elastic bounds and find a solution that minimizes f subject to the non-elastic constraints. This option is useful if there are some 'soft' constraints that you would like to ignore if the constraints are infeasible.
1	Use a composite objective defined with γ determined by the value of **Elastic Weight**. This value is intended to be used in conjunction with **Elastic Mode** $= 2$.
2	Include only the elastic variables in the composite objective. The elastics are weighted by $\gamma = 1$. This choice minimizes the violations of the elastic variables at the expense of possibly increasing the true objective. This option can be used to find a point that minimizes the sum of the violations of a subset of constraints determined by the argument **helast**.

Elastic Weight – double r Default $= 1.0$

This keyword defines the value of γ in the composite objective.

At each iteration of elastic mode, the composite objective is defined to be

$$\text{minimize } \sigma f(x) + \gamma \text{ (sum of infeasibilities);}$$

where $\sigma = 1$ for **Minimize**, $\sigma = -1$ for **Maximize**, and f is the current objective value.

Note that the effect of γ is *not* disabled once a feasible iterate is obtained.

Expand Frequency – Integer i Default $= 10000$

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

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 argument **Feasibility Tolerance** is δ. Over a period of i iterations, the feasibility tolerance actually used by nag_opt_sparse_convex_qp_solve (e04nqc) (i.e., the *working* feasibility tolerance) increases from 0.5δ to δ (in steps of $0.5\delta/i$).

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

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 – Integer i Default $= 100(\mathrm{LP})$ or $50(\mathrm{QP})$

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. Higher values of i may be more efficient on problems that are extremely sparse and well scaled. 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 **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 – double r Default $= 10^{-6}$

A *feasible problem* is one in which all variables satisfy their upper and lower bounds to within the absolute tolerance r. (This includes slack variables. Hence, the general constraints are also satisfied to within r.)

nag_opt_sparse_convex_qp_solve (e04nqc) 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 if **sinf** is not small and you have not asked nag_opt_sparse_convex_qp_solve (e04nqc) to minimize the violations of the elastic variables (i.e., you have not specified **Elastic Objective** $= 2$, there may be other points that have a *significantly smaller sum of infeasibilities*. nag_opt_sparse_convex_qp_solve (e04nqc) will not attempt to find the solution that minimizes the sum unless **Elastic Objective** $= 2$.

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

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

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

Iteration Limit – Integer i Default $= \max(10000, m)$
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**
Nolist

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

LU Factor Tolerance – double r_1 Default $= 3.99$
LU Update Tolerance – double r_2 Default $= 3.99$

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. They must satisfy r_1, $r_2 \geq 1.0$.

For large and relatively dense problems, $r_1 = 10.0$ or 5.0 (say) may give a useful improvement in stability without impairing sparsity to a serious degree.

For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a sub-matrix of the form

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

one should set both r_1 and r_2 to values in the range $1.0 \leq r_i < 4.0$.

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

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

Maximize Default $=$ **Minimize**
Minimize

This option specifies the required direction of the optimization. It applies to both linear and nonlinear terms (if any) in the objective function. Note that if two problems are the same except that one minimizes $f(x)$ and the other maximizes $-f(x)$, their solutions will be the same but the signs of the dual variables π_i and the reduced gradients d_j (see Section 10.3) will be reversed.

New Basis File – Nag_FileID i_1 Default $= 0$
Backup Basis File – Nag_FileID i_2 Default $= 0$
Save Frequency – Integer i_3 Default $= 100$

(See Section 11.1 for a description of **Nag_FileID**.)

New Basis File and **Backup Basis File** sometimes referred to as basis maps. They contain the most compact representation of the state of each variable. They are intended for restarting the solution of a problem at a point that was reached by an earlier run. For non-trivial problems, it is advisable to save basis maps at the end of a run, in order to restart the run if necessary.

If **New Basis File** > 0, a basis map will be saved on file **New Basis File** every i_3th iteration, where i_3 is the **Save Frequency**. The first record of the file will contain the word `PROCEEDING` if the run is still in progress. A basis map will also be saved at the end of a run, with some other word indicating the final solution status.

If **Backup Basis File** > 0, **Backup Basis File** is intended as a safeguard against losing the results of a long run. Suppose that a **New Basis File** is being saved every 100 (**Save Frequency**) iterations, and that nag_opt_sparse_convex_qp_solve (e04nqc) is about to save such a basis at iteration 2000. It is conceivable that the run may be interrupted during the next few milliseconds (in the middle of the save). In this case the basis file will be corrupted and the run will have been essentially wasted.

To eliminate this risk, both a **New Basis File** and a **Backup Basis File** may be specified. The following would be suitable for the above example:

```
Backup Basis FileID1
New Basis FileID2
```

where `FileID1` and `FileID2` are returned by nag_open_file (x04acc).

The current basis will then be saved every 100 iterations, first on `FileID2` and then immediately on `FileID1`. If the run is interrupted at iteration 2000 during the save on `FileID2`, there will still be a usable basis on `FileID1` (corresponding to iteration 1900).

Note that a new basis will be saved in **New Basis File** at the end of a run if it terminates normally, but it will not be saved in **Backup Basis File**. In the above example, if an optimum solution is found at iteration 2050 (or if the iteration limit is 2050), the final basis on `FileID2` will correspond to iteration 2050, but the last basis saved on `FileID1` will be the one for iteration 2000.

A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (1999).

Old Basis File – Nag_FileID i Default $= 0$

(See Section 11.1 for a description of **Nag_FileID**.)

If **Old Basis File** > 0, the basis maps information will be obtained from the file associated with ID i. A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (1999). The file will usually have been output previously as a **New Basis File** or **Backup Basis File**.

The file will not be acceptable if the number of rows or columns in the problem has been altered.

Optimality Tolerance – double r Default $= 10^{-6}$

This is used to judge the size of the reduced gradients $d_j = g_j - \pi a_j$, where g_j is the jth component of the gradient, a_j is the associated column of the constraint matrix $(A \quad -I)$, and π is the set of dual variables.

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

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

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

In the above tests, $\|\pi\|$ 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 quantity $\|\pi\|$ actually used is defined by

$$\|\pi\| = \max\left\{\sigma\sqrt{m}, 1\right\}, \quad \text{where } \sigma = \sum_{i=1}^{m} |\pi_i|j$$

so that only large scale factors are allowed for.

If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against $0.01r$.

Partial Price – Integer i Default $= 10(\mathrm{LP})$ or $1(\mathrm{QP})$

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

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

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

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

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

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

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

Print File – Nag_FileID i Default $= 0$

(See Section 11.1 for a description of **Nag_FileID**.)

If **Print File** > 0, the following information is output to **Print File** during the solution of each problem:

– a listing of the optional arguments;

– some statistics about the problem;

– the amount of storage available for the LU factorization of the basis matrix;

– notes about the initial basis resulting from a Crash procedure or a basis file;

– the iteration log;

– basis factorization statistics;

– the exit fail condition and some statistics about the solution obtained;

– the printed solution, if requested.

The last four items are described in Sections 8 and 12. Further brief output may be directed to the **Summary File**.

Print Frequency – Integer i Default $= 100$

If $i > 0$, one line of the iteration log will be printed every ith iteration. A value such as $i = 10$ is suggested for those interested only in the final solution.

Print Level – Integer i Default $= 1$

This controls the amount of printing produced by nag_opt_sparse_convex_qp_solve (e04nqc) as follows.

i	**Meaning**
0	No output except error messages. If you want to suppress all output, set **Print File** $= 0$.
$= 1$	The set of selected options, problem statistics, summary of the scaling procedure, information about the initial basis resulting from a crash or a basis file. a single line of output at each iteration (controlled by **Print Frequency**), and the exit condition with a summary of the final solution.
≥ 10	Basis factorization statistics.

Punch File – Nag_FileID i_1 Default $= 0$
Insert File – Nag_FileID i_2 Default $= 0$

(See Section 11.1 for a description of **Nag_FileID**.)

These files provide compatibility with commercial mathematical programming systems. The **Punch File** from a previous run may be used as an **Insert File** for a later run on the same problem. A full description of information recorded in **Insert File** and **Punch File** is given in Gill *et al.* (1999).

If **Insert File** > 0, the final solution obtained will be output to file **Punch File**. For linear programs, this format is compatible with various commercial systems.

If **Punch File** > 0, the **Insert File** containing basis information will be read. The file will usually have been output previously as a **Punch File**. The file will not be accessed if **Old Basis File** is specified.

Scale Option – Integer i Default $= 2$
Scale Tolerance – double r Default $= 0.9$

Three scale options are available as follows:

i	**Meaning**
0	No scaling. This is recommended if it is known that x and the constraint matrix never have very large elements (say, larger than 1000).
1	The constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer (1982)). This will sometimes improve the performance of the solution procedures.
2	The constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of $\begin{pmatrix} A & -I \end{pmatrix}$ that are fixed or have positive lower bounds or negative upper bounds.

Scale Tolerance affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest non-zero coefficients in each column:

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

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

Solution File – Nag_FileID $\quad i \quad$ Default $= 0$

(See Section 11.1 for a description of **Nag_FileID**.)

If **Solution File** > 0, the final solution will be output to file **Solution File** (whether optimal or not).

To see more significant digits in the printed solution, it will sometimes be useful to make **Solution File** refer to the system **Print File**.

Summary File – Nag_FileID $\quad i_1 \quad$ Default $= 0$
Summary Frequency – Integer $\quad i_2 \quad$ Default $= 100$

(See Section 11.1 for a description of **Nag_FileID**.)

If **Summary File** > 0, a brief log will be output to file **Summary File**, including one line of information every i_2th iteration. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. (If something looks wrong, the run can be manually terminated.) Further details are given in Section 12.

Superbasics Limit – Integer $\quad i \quad$ Default $= \min(500, n_H + 1, n)$

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

For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than m, the number of general constraints.) The default value of i is therefore 1.

For quadratic problems, the number of degrees of freedom is often called the 'number of independent variables'.

Normally, i need not be greater than **ncolh** $+ 1$, where **ncolh** is the number of leading non-zero columns of H, n_H.

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

Suppress Parameters

Normally nag_opt_sparse_convex_qp_solve (e04nqc) prints the options file as it is being read, and then prints a complete list of the available keywords and their final values. The **Suppress Parameters** option tells nag_opt_sparse_convex_qp_solve (e04nqc) not to print the full list.

Timing Level – Integer i Default $= 0$

If $i > 0$, some timing information will be output to the **Print File**.

Unbounded Step Size – double 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.

12 Description of Monitoring Information

This section describes the intermediate printout and final printout which constitutes the monitoring information produced by nag_opt_sparse_convex_qp_solve (e04nqc). (See also the description of the optional arguments **Print File** and **Print Level** in Section 11.2.) The level of printed output can be controlled by you.

When **Print Level** > 20 and **Print File** > 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by **Print 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 option **LU Factor Tolerance** (see Section 11.2).

Label	**Description**
`Factorize`	is the factorization count.
`Demand`	is a code giving the reason for the present factorization as follows: **Code** **Meaning** 0 First LU factorization. 1 The number of updates reached the value of the optional argument **Factorization Frequency** (see Section 11.2). 2 The number of non-zeros in the updated factors has increased significantly. 7 Not enough storage to update factors. 10 Row residuals too large (see the description for the option **Check Frequency** in Section 11.2). 11 Ill-conditioning has caused inconsistent results.
`Iteration`	is the iteration count.
`Infeas`	the number of infeasibilities at the *start* of the previous iteration.
`Objective`	if `Infeas` > 0, this is the Sum of Infeasibilities at the *start* of the previous iteration.
`Nonlinear`	is the number of nonlinear variables in the current basis B (not printed if B_S is factorized). If `Infeas` $= 0$, this is the value of the objective function after the previous iteration.
`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 non-zeros in B (not printed if B_S is factorized)
`Density`	is the percentage non-zero density of B (not printed if B_S is factorized). More precisely, `Density` $= 100 \times$ `Elems`$/(m \times m)$, where m is the number of rows in the problem ($m =$ `Linear` $+$ `Slacks`).
`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.
`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 non-zeros 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 non-zeros in L.
`lenU`	is the number of non-zeros in U.
`Increase`	is the percentage increase in the number of non-zeros in L and U relative to the number of non-zeros in B. More precisely, $\mathtt{Increase} = 100 \times (\mathtt{lenL} + \mathtt{lenU} - \mathtt{Elems})/\mathtt{Elems}$.
`m`	is the number of rows in the problem. Note that $\mathtt{m} = \mathtt{Ut} + \mathtt{Lt} + \mathtt{bp}$.
`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 argument **LU Factor Tolerance** (see Section 11.2).
`Bmax`	is the maximum non-zero element in B (not printed if B_S is factorized).
`BSmax`	is the maximum non-zero element in B_S (not printed if B is factorized).
`Umax`	is the maximum non-zero 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 **LU Factor Tolerance** to some value nearer unity. `Umax` is not printed if B_S is factorized.
`Umin`	is the magnitude of the smallest diagonal element of PUQ (not printed if B_S is factorized).
`Growth`	is the value of the ratio `Umax/Bmax`, which should not be too large. Providing `Lmax` is not large (say, < 10.0), the ratio $\max(\mathtt{Bmax}, \mathtt{Umax})/\mathtt{Umin}$ is an estimate of the condition number of B. If this number is extremely large, the basis is nearly singular and some numerical difficulties might occur. (However, an effort is made to avoid near-singularity by using slacks to replace columns of B that would have made `Umin` extremely small and the modified basis is refactorized.) `Growth` is not printed if B_S is factorized.
`Lt`	is the number of triangular columns of B at the left of L.
`bp`	is the size of the 'bump' or block to be factorized nontrivially after the triangular rows and columns of B have been removed.
`d2`	is the number of columns remaining when the density of the basis matrix being factorized has reached 0.6.

When **Print Level** > 20 and **Print File** > 0, the following lines of intermediate printout (< 120 characters) are produced on the unit number specified by **Print File** whenever **start** $=$ **Nag_Cold** (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.

Label	**Description**
`Slacks`	is the number of slacks selected initially.

`Free cols`	is the number of free columns in the basis, including those whose bounds are rather far apart.
`Preferred`	is the number of 'preferred' columns in the basis (i.e., **hs**[j] $= 3$ for some $j \leq n$). It will be a subset of the columns for which **hs**[j] $= 3$ was specified.
`Unit`	is the number of unit columns in the basis.
`Double`	is the number of double columns in the basis.
`Triangle`	is the number of triangular columns in the basis.
`Pad`	is the number of slacks used to pad the basis (to make it a non-singular triangle).

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

Label	**Description**
`Name`	gives the name for the problem (blank if none).
`Status`	gives the exit status for the problem (i.e., `Optimal soln`, `Weak soln`, `Unbounded`, `Infeasible`, `Excess itns`, `Error condn` or `Feasble soln`) followed by details of the direction of the optimization (i.e., `(Min)` or `(Max)`).
`Objective`	gives the name of the free row for the problem (blank if 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).

At the end of a run, the final solution will be output to the **Print File**. Some header information appears first to identify the problem and the final state of the optimization procedure. A ROWS section and a COLUMNS section then follow, giving one line of information for each row and column.

The ROWS section

The general constraints take the form $l \leq Ax \leq u$. The ith constraint is therefore of the

$$\infty \leq \nu_i^{\mathrm{T}} x \leq \beta,$$

where ν_i is the ith row of A.

Internally, the constraints take the form $Ax - s = 0$, where s is the set of slack variables (which happen to satisfy the bounds $l \leq s \leq u$). For the ith constraint it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. A '.' is printed for any numerical value that is exactly zero.

Label	**Description**
`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 v_i.
`State`	the state of vi (the state of s_i relative to the bounds α and β. The various states possible are as follows:

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

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

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

`FR` s_i is nonbasic and currently zero, even though it is free to take any value between its bounds α and β.

`BS` s_i is basic.

`SBS` s_i is superbasic.

A key is sometimes printed before `State` to give some additional information about the state of a variable. Note that unless the optional argument **Scale Option** $= 0$ (see Section 11.2) is specified, the tests for assigning a key are applied to the variables of the scaled problem.

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

`D` *Degenerate*. The variable is basic or superbasic, but it is equal 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 argument **Feasibility Tolerance** (see Section 11.2).

`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 argument **Optimality Tolerance** (see Section 11.2), 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 (the ith element of $A^{\mathrm{T}}x$).

`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 Limit` is α, the lower bound specified for the variable s_i. `None` indicates that $\mathbf{bl}[j] \leq -bigbnd$.

`Upper Bound` is β, the upper bound specified for the variable s_i. `None` indicates that $\mathbf{bu}[j] \geq bigbnd$.

`Dual Activity` is the value of the dual variable π_i (the Lagrange-multiplier for v_i; see Section 10.3). For FP problems, π_i is set to zero.

`i` gives the index i of the ith row.

The COLUMNS section

Let the jth component of x be the variable x_j and assume that it satisfies the bounds $\alpha \leq x_j \leq \beta$. A '.' is printed for any numerical value that is exactly zero.

Label	Description
`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`	the state of x_j relative to the bounds α and β. The various states possible 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 and currently zero, even though it is free to take any value between its bounds α and β.

`BS` x_j is basic.

`SBS` x_j is superbasic.

A key is sometimes printed before `State` to give some additional information about the state of a variable. Note that unless the optional argument **Scale Option** $= 0$ (see Section 11.2) is specified, the tests for assigning a key are applied to the variables of the scaled problem.

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

`D` *Degenerate*. The variable is basic or superbasic, but it is equal 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 argument **Feasibility Tolerance** (see Section 11.2).

`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 argument **Optimality Tolerance** (see Section 11.2), 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 **bl**[j] $\leq -bigbnd$.

`Upper Bound` is the upper bound specified for the variable. `None` indicates that **bu**[j] $\geq bigbnd$.

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

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

Note: 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 `dj` will be reversed.

The SOLUTION file

If **Solution File** > 0, the information contained in a printed solution may also be output to the relevant file (which may be the **Print File** if so desired). Infinite Upper and Lower limits appear as 10^{20}rather than None. Again, the maximum line length is 111 characters.

A **Solution File** is intended to be read from disk by a self-contained program that extracts and saves certain values as required for possible further computation. Typically the first 14 lines would be ignored. The end of the ROWS section is marked by a line that starts with a 1 and is otherwise blank. If this and the next 4 lines are skipped, the COLUMNS section can then be read under the same format.

The SUMMARY file

If **Summary File** > 0, certain brief information will be output to file. A disk file should be used to retain a concise log of each run if desired. (A **Summary File** is more easily perused than the associated **Print File**).

The following information is included:

1. The Begin line from the optional arguments file, if used;

2. The basis file loaded, if any;
3. The status of the solution after each basis factorization (whether feasible; the objective value; the number of function calls so far);
4. The same information every kth iteration, where k is the specified **Summary Frequency** (see Section 11.2);
5. Warnings and error messages;
6. The exit condition and a summary of the final solution.

Item 4. is preceded by a blank line, but item 5. is not.

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, **names**$[j-1]$ replaced by **names**$[n+j-1]$, **bl**$[j-1]$ and **bu**$[j-1]$ are replaced by **bl**$[n+j-1]$ and **bu**$[n+j-1]$ 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.