nag_opt_qp (e04nfc)

1. Purpose

nag_opt_qp solves general quadratic programming problems. It is not intended for large sparse problems.

2. Specification

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

void nag_opt_qp(Integer n, Integer nclin, double a[], Integer tda, double bl[],
                double bu[], double cvec[], double h[], Integer tdh,
                void (*qphess)(Integer n, Integer jthcol, double h[], Integer tdh,
                               double x[], double hx[], Nag_Comm *comm),
                double x[], double *objf, Nag_E04_Opt *options,
                Nag_Comm *comm, NagError *fail)
```

3. Description

nag_opt_qp is designed to solve a class of quadratic programming problems stated in the following general form:

$$\underset{x \in R^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad l \le \left\{ \begin{matrix} x \\ Ax \end{matrix} \right\} \le u,$$

where A is an m_{lin} by n matrix and $f(x)$ may be specified in a variety of ways depending upon the particular problem to be solved. The available forms for $f(x)$ are listed in Table 1 below, in which the prefixes FP, LP and QP stand for 'feasible point', 'linear programming' and 'quadratic programming' respectively and c is an n element vector.

Problem Type	$f(x)$	**Matrix** H
FP	Not applicable	Not applicable
LP	c^Tx	Not applicable
QP1	$\frac{1}{2}x^THx$	symmetric
QP2	$c^Tx + \frac{1}{2}x^THx$	symmetric
QP3	$\frac{1}{2}x^TH^THx$	m by n upper trapezoidal
QP4	$c^Tx + \frac{1}{2}x^TH^THx$	m by n upper trapeziodal

Table 1

For problems of type FP a feasible point with respect to a set of linear inequality constraints is sought. The default problem type is QP2, other objective functions are selected by using the optional parameter **prob** (see Section 8.2).

The constraints involving A are called the *general* constraints. Note that upper and lower bounds are specified for all the variables and for all the general constraints. An *equality* constraint can be specified by setting $l_i = u_i$. If certain bounds are not present, the associated elements of l or u can be set to special values that will be treated as $-\infty$ or $+\infty$. (See the description of the optional parameter **inf_bound** in Section 8.2.)

The defining feature of a quadratic function $f(x)$ is that the second-derivative matrix $\nabla^2 f(x)$ (the *Hessian matrix*) is constant. For the LP case, $\nabla^2 f(x) = 0$; for QP1 and QP2, $\nabla^2 f(x) = H$; and for QP3 and QP4, $\nabla^2 f(x) = H^TH$. If H is defined as the zero matrix, nag_opt_qp will solve the resulting linear programming problem; however, this can be accomplished more efficiently by setting the optional parameter **prob** = **Nag_LP**, or by using nag_opt_lp (e04mfc).

The user must supply an initial estimate of the solution.

In the QP case, the user may supply H either *explicitly* as an m by n matrix, or *implicitly* in a C function that computes the product Hx for any given vector x. An example of such a function is included in the example program in Section 6. There is no restriction on H apart from symmetry.

In general, a successful run of nag_opt_qp will indicate one of three situations: (i) a minimizer has been found; (ii) the algorithm has terminated at a so-called *dead-point*; or (iii) the problem has no bounded solution. If a minimizer is found, and H is positive-definite or positive semi-definite, nag_opt_qp will obtain a global minimizer; otherwise, the solution will be a *local minimizer* (which may or may not be a global minimizer). A dead-point is a point at which the necessary conditions for optimality are satisfied but the sufficient conditions are not. At such a point, a feasible direction of decrease may or may not exist, so that the point is not necessarily a local solution of the problem. Verification of optimality in such instances requires further information, and is in general an NP-hard problem (see Pardalos and Schnitger (1988)). Termination at a dead-point can occur only if H is not positive-definite. If H is positive semi-definite, the dead-point will be a *weak minimizer* (i.e., with a unique optimal objective value, but an infinite set of optimal x).

Details about the algorithm are described in Section 7, but it is not necessary to read this more advanced section before using nag_opt_qp.

4. Parameters

n

Input: n, the number of variables.
Constraint: $\mathbf{n} > 0$.

nclin

Input: m_{lin}, the number of general linear constraints.
Constraint: $\mathbf{nclin} \geq 0$.

a[nclin][tda]

Input: the ith row of **a** must contain the coefficients of the ith general linear constraint (the ith row of A), for $i = 1, 2, \ldots, m_{lin}$. If $\mathbf{nclin} = 0$ then the array **a** is not referenced.

tda

Input: the second dimension of the array **a** as declared in the function from which nag_opt_qp is called.
Constraint: $\mathbf{tda} \geq \mathbf{n}$ if $\mathbf{nclin} > 0$.

bl[n+nclin]
bu[n+nclin]

Input: **bl** must contain the lower bounds and **bu** the upper bounds, for all the constraints in the following order. The first n elements of each array must contain the bounds on the variables, and the next m_{lin} elements the bounds for the general linear constraints (if any). To specify a non-existent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}[j] \leq -\mathbf{inf\_bound}$, and to specify a non-existent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}[j] \geq \mathbf{inf\_bound}$; here **inf_bound** is the optional parameter **options.inf_bound**, whose default value is 10^{20} (see Section 8.2). To specify the jth constraint as an *equality*, set $\mathbf{bl}[j] = \mathbf{bu}[j] = \beta$, say, where $|\beta| < \mathbf{inf\_bound}$.
Constraints:

$\mathbf{bl}[j] \leq \mathbf{bu}[j]$, for $j = 0, 1, \ldots, \mathbf{n}+\mathbf{nclin}-1$,

$|\beta| < \mathbf{inf\_bound}$ when $\mathbf{bl}[j] = \mathbf{bu}[j] = \beta$.

cvec[n]

Input: the coefficients of the explicit linear term of the objective function when the problem is of type **Nag_LP**, **Nag_QP2** and **Nag_QP4**. The default problem type is **Nag_QP2** corresponding to QP2 described in Section 3; other problem types can be specified using the optional parameter **prob**; see Section 8.2.

If the problem is of type **Nag_FP**, **Nag_QP1** or **Nag_QP3**, **cvec** is not referenced and therefore a NULL pointer may be given.

h[n][tdh]

Input: **h** may be used to store the quadratic term H of the QP objective function if desired. The elements of **h** are accessed only by the function **qphess**; thus **h** is not accessed if the problem is of type **Nag_FP** or **Nag_LP**. The number of rows of H is denoted by m, its default value is equal to n. (The optional parameter **hrows** may be used to specify a value of $m < n$; see Section 8.2.)

If the problem is of type **Nag_QP1** or **Nag_QP2**, the first m rows and columns of **h** must contain the leading m by m rows and columns of the symmetric Hessian matrix. Only the diagonal and upper triangular elements of the leading m rows and columns of **h** are referenced. The remaining elements need not be assigned.

For problems **Nag_QP3** and **Nag_QP4**, the first m rows of **h** must contain an m by n upper trapezoidal factor of the Hessian matrix. The factor need not be of full rank, i.e., some of the diagonals may be zero. However, as a general rule, the larger the dimension of the leading non-singular sub-matrix of H, the fewer iterations will be required. Elements outside the upper trapezoidal part of the first m rows of H are assumed to be zero and need not be assigned.

In some cases, the user need not use **h** to store H explicitly (see the specification of function **qphess** below).

tdh

Input: the second dimension of the array **h** as declared in the function from which nag_opt_qp is called.
Constraint: **tdh** $\geq$ **n** or at least the value of the optional parameter **hrows** if it is set.

qphess

In general, the user need not provide a version of **qphess**, because a 'default' function is included in the NAG C Library. If the default function is required then the NAG defined null void function pointer, `NULLFN`, should be supplied in the call to nag_opt_qp. The algorithm of nag_opt_qp requires only the product of H and a vector x; and in some cases the user may obtain increased efficiency by providing a version of **qphess** that avoids the need to define the elements of the matrix H explicitly.

qphess is not referenced if the problem is of type **Nag_FP** or **Nag_LP**, in which case **qphess** should be replaced by `NULLFN`.

The specification of **qphess** is:

```
void qphess(Integer n, Integer jthcol, double h[], Integer tdh, double x[],
            double hx[], Nag_Comm *comm)
```

n

Input: n, the number of variables.

jthcol

Input: **jthcol** specifies whether or not the vector x is a column of the identity matrix. If **jthcol** $= j > 0$, then the vector x is the jth column of the identity matrix, and hence Hx is the jth column of H, which can sometimes be computed very efficiently and **qphess** may be coded to take advantage of this. However special code is not necessary because x is always stored explicitly in the array **x**. If **jthcol** $= 0$, x has no special form.

h[n*tdh]

Input: the matrix H of the QP objective function.

The matrix element H_{ij} is stored in `h[(`i`-1)*tdh + `j`-1]` for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$. In some situations, it may be desirable to compute Hx without accessing **h** – for example, if H is sparse or has special structure. (This is illustrated in the function `qphess1` in the example program in Section 6.) The parameters **h** and **tdh** may then refer to any convenient array.

tdh

Input: the second dimension of the array **h** in the calling program.

x[n]

Input: the vector x.

hx[n]

Output: the product Hx.

comm

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

flag – Integer

Input: **comm->flag** contains a non-negative number.

Output: if **qphess** resets **comm->flag** to some negative number nag_opt_qp will terminate immediately with the error indicator **NE_USER_STOP**. If **fail** is supplied to nag_opt_qp, **fail.errnum** will be set to the user's setting of **comm->flag**.

first – Boolean

Input: will be set to **TRUE** on the first call to **qphess** and **FALSE** for all subsequent calls.

nf – Integer

Input: the number of calls made to **qphess** including the current one.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

Before calling nag_opt_qp these pointers may be allocated memory by the user and initialized with various quantities for use by **qphess** when called from nag_opt_qp.

Note: **qphess** should be tested separately before being used in conjunction with nag_opt_qp. The input arrays **h** and **x** must **not** be changed within **qphess**.

x[n]

Input: an initial estimate of the solution.

Output: the point at which nag_opt_qp terminated. If **fail.code** = **NE_NOERROR**, **NW_DEAD_POINT**, **NW_SOLN_NOT_UNIQUE** or **NW_NOT_FEASIBLE**, **x** contains an estimate of the solution.

objf

Output: the value of the objective function at x if x is feasible, or the sum of infeasibilities at x otherwise. If the problem is of type **Nag_FP** and x is feasible, **objf** is set to zero.

options

Input/Output: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_qp. These structure members offer the means of adjusting some of the parameter values of the algorithm and on output will supply further details of the results. A description of the members of **options** is given below in Section 8. Some of the results returned in **options** can be used by nag_opt_qp to perform a 'warm start' if it is re-entered (see the member **start** in Section 8.2).

If any of these optional parameters are required then the structure **options** should be declared and initialized by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_opt_qp. However, if the optional parameters are not required the NAG defined null pointer, `E04_DEFAULT`, can be used in the function call.

comm

Input/Output: structure containing pointers for user communication with user-supplied functions; see the above description of **qphess** for details. If the user does not need to make use of this communication feature the null pointer `NAGCOMM_NULL` may be used in the call to nag_opt_qp; **comm** will then be declared internally for use in calls to user-supplied functions.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

Users are recommended to declare and initialize **fail** and set **fail.print** = **TRUE** for this function. nag_opt_qp returns with **fail.code** = **NE_NOERROR** if x is a strong local minimizer, i.e., the reduced gradient is negligible, the Lagrange multipliers are optimal and H_r is positive semi-definite.

4.1. Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled by the user with the structure member **options.print_level** (see Section 8.2). The default print level of **Nag_Soln_Iter** provides a single line of output at each iteration and the final result. This section describes the default printout produced by nag_opt_qp.

The convention for numbering the constraints in the iteration results is that indices 1 to n refer to the bounds on the variables, and indices $n+1$ to $n+m_{lin}$ refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

The single line of intermediate results output on completion of each iteration gives:

`Itn` is the iteration count.

`Jdel` is the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` is the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` is the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. During the optimality phase, the step can be greater than 1.0 only if the reduced Hessian is not positive-definite.

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

`Sinf/Obj` is the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If x is feasible, `Obj` 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. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Bnd` the number of simple bound constraints in the current working set.

`Lin` the number of general linear constraints in the current working set.

`Nart` the number of artificial constraints in the working set. At the start of the optimality phase, `Nart` provides an estimate of the number of nonpositive eigenvalues in the reduced Hessian.

`Nrz` the dimension of the subspace in which the objective function is currently being minimized. The value of `Nrz` is the number of variables minus the number of constraints in the working set; i.e., $\texttt{Nrz} = n - (\texttt{Bnd} + \texttt{Lin} + \texttt{Nart})$.

`Norm Gz` the Euclidean norm of the reduced gradient. During the optimality phase, this norm will be approximately zero after a unit step.

The printout of the final result consists of:

`Varbl` the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State`	the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the feasibility tolerance, `State` will be `++` or `--` respectively.
`Value`	the value of the variable at the final iteration.
`Lower bound`	the lower bound specified for the variable. (`None` indicates that $\mathbf{bl}[j-1] \le -\mathbf{inf\_bound}$.)
`Upper bound`	the upper bound specified for the variable. (`None` indicates that $\mathbf{bu}[j-1] \ge \mathbf{inf\_bound}$.)
`Lagr mult`	the value of the Lagrange multiplier for the associated bound constraint. This will be zero if `State` is `FR`. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.
`Residual`	the difference between the variable `Value` and the nearer of its bounds $\mathbf{bl}[j-1]$ and $\mathbf{bu}[j-1]$.

The meaning of the printout for general constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', and with the following change in the heading:

`LCon`	the name (`L`) and index j, for $j = 1, 2, \ldots, m_{lin}$ of the constraint.

5. Comments

A list of possible error exits and warnings from nag_opt_qp is given in Section 9. Scaling and accuracy are considered in Section 10.

6. Example 1

This example problem is taken from Bunch and Kaufman (1980) and involves the minimization of the quadratic function $f(x) = c^T x + \frac{1}{2} x^T H x$, where

$$c = (7.0,\ 6.0,\ 5.0,\ 4.0,\ 3.0,\ 2.0,\ 1.0,\ 0.0)^T$$

$$H = \begin{pmatrix} 1.69 & 1.00 & 2.00 & 3.00 & 4.00 & 5.00 & 6.00 & 7.00 \\ 1.00 & 1.69 & 1.00 & 2.00 & 3.00 & 4.00 & 5.00 & 6.00 \\ 2.00 & 1.00 & 1.69 & 1.00 & 2.00 & 3.00 & 4.00 & 5.00 \\ 3.00 & 2.00 & 1.00 & 1.69 & 1.00 & 2.00 & 3.00 & 4.00 \\ 4.00 & 3.00 & 2.00 & 1.00 & 1.69 & 1.00 & 2.00 & 3.00 \\ 5.00 & 4.00 & 3.00 & 2.00 & 1.00 & 1.69 & 1.00 & 2.00 \\ 6.00 & 5.00 & 4.00 & 3.00 & 2.00 & 1.00 & 1.69 & 1.00 \\ 7.00 & 6.00 & 5.00 & 4.00 & 3.00 & 2.00 & 1.00 & 1.69 \end{pmatrix}$$

subject to the bounds

$$\begin{aligned} -1.0 &\le x_1 \le 1.0 \\ -2.1 &\le x_2 \le 2.0 \\ -3.2 &\le x_3 \le 3.0 \\ -4.3 &\le x_4 \le 4.0 \\ -5.4 &\le x_5 \le 5.0 \\ -6.5 &\le x_6 \le 6.0 \\ -7.6 &\le x_7 \le 7.0 \\ -8.7 &\le x_8 \le 8.0 \end{aligned}$$

and the general constraints

$$
\begin{aligned}
-x_1 + x_2 &\geq -1.00\\
- x_2 + x_3 &\geq -1.05\\
- x_3 + x_4 &\geq -1.10\\
- x_4 + x_5 &\geq -1.15\\
- x_5 + x_6 &\geq -1.20\\
- x_6 + x_7 &\geq -1.25\\
- x_7 + x_8 &\geq -1.30
\end{aligned}
$$

The initial point is

$$x_0 = (-1.0,\ -2.0,\ -3.0,\ -4.0,\ -5.0,\ -6.0,\ -7.0,\ -8.0)^T.$$

The computed solution is

$$x^* = (-1.0,\ -2.0,\ -3.05,\ -4.15,\ -5.3,\ 6.0,\ 7.0,\ 8.0)^T.$$

Four bound constraints and four general constraints are active at the solution.

This example shows the simple use of nag_opt_qp where default values are used for all optional parameters. An example showing the use of optional parameters is given in Section 13. There is one example program file, the main program of which calls both examples. The C functions for the main program and Example 1 are given below. In Example 1 the problem is solved twice, first with the Hessian explicit and Hx calculated by nag_opt_qp and then with the Hessian implicit and Hx formed by a user supplied function, `qphess1`.

6.1. Program Text

```
/* nag_opt_qp (e04nfc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 6 revised, 2000.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <nage04.h>

static void qphess1(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm);
static void qphess2(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm);
static void qphess3(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm);
static void ex1(void);
static void ex2(void);

#define MAXN 10
#define MAXLIN 7
#define MAXBND MAXN+MAXLIN

main(void)
{
  /* Two examples are called, ex1() uses the
   * default settings to solve a problem while
   * ex2() solves another problem with some
   * of the optional parameters set by the user.
   */

  Vprintf("e04nfc Example Program Results.\n");
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}
```

```
static void ex1()
{
  double a[MAXLIN][MAXN], h[MAXN][MAXN];
  double x[MAXN], cvec[MAXN];
  double bl[MAXBND], bu[MAXBND];
  double bigbnd, objf;
  Integer i, j, n, nclin, tda, tdh;
  static NagError fail;

  Vprintf("\nExample 1: default options used.\n");

  fail.print = TRUE;

  /* Define the problem. This example is due to Bunch and Kaufman,
   * ‘A computational method for the indefinite quadratic programming
   * problem ’, Linear Algebra and its Applications, 34, 341-370 (1980).
   *
   * h  = the QP Hessian matrix.
   * a  = the general constraint matrix.
   * bl = the lower bounds on x and A*x.
   * bu = the upper bounds on x and A*x.
   * x  = the initial estimate of the solution.
   *
   * Set the actual problem dimensions.
   * n      = the number of variables.
   * nclin  = the number of general linear constraints (may be 0).
   */
  n = 8;
  nclin = 7;
  tda = MAXN;
  tdh = MAXN;

  /* Define the value used to denote ‘‘infinite’’ bounds. */
  bigbnd = 1.0e20;

  for (i = 0; i < nclin; ++i)
    for (j = 0; j < n; ++j)
      a[i][j] = 0.0;

  for (i = 0; i < nclin; ++i)
    {
      a[i][i] = -1.0;
      a[i][i+1] = 1.0;
      bl[n + i] = -1.0 - 0.05*(double)i;
      bu[n + i] = bigbnd;
    }

  for (j = 0; j < n; ++j)
    {
      bl[j] = -(double)(j+1) - 0.1*(double)(j);
      bu[j] = (double)(j+1);
      cvec[j] = (double)(7 - j);
    }

  for (i = 0; i < n; ++i)
    {
      for (j = i+1; j < n; ++j)
        h[i][j] = (double)(ABS(i-j));
      h[i][i] = 1.69;
    }

  /* Set the initial estimate of the solution. */
  x[0] = -1.0;
  x[1] = -2.0;
  x[2] = -3.0;
  x[3] = -4.0;
  x[4] = -5.0;
  x[5] = -6.0;
  x[6] = -7.0;
  x[7] = -8.0;
```

```
  /* Solve the QP problem. */
  e04nfc(n, nclin, (double *)a, tda, bl, bu, cvec, (double *)h, tdh,
         NULLFN, x, &objf, E04_DEFAULT, NAGCOMM_NULL, &fail);

  if (fail.code == NE_NOERROR)
    {
      Vprintf("Re-solve problem with the Hessian defined by function qphess1.\n");

      /* Set a new initial estimate of the solution. */
      x[0] = -1.0;
      x[1] = 12.0;
      x[2] = -3.0;
      x[3] = 14.0;
      x[4] = -5.0;
      x[5] = 16.0;
      x[6] = -7.0;
      x[7] = 18.0;

      /* Solve the QP problem. */
      e04nfc(n, nclin, (double *)a, tda, bl, bu, cvec, (double *)0, tdh,
             qphess1, x, &objf, E04_DEFAULT, NAGCOMM_NULL, &fail);
    }
  if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);
}       /* ex1 */


static void qphess1(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm)
{
  /* qphess1 computes the vector  Hx = (H)*x  for some matrix H
   * that defines the Hessian of the required QP problem.
   *
   * In this version qphess the Hessian matrix is implicit.
   * The array h[] is not accessed. There is no special coding
   * for the case jthcol > 0
   */
  Integer i, j;
  double sum;

  for (i = 0; i < n; ++i)
    {
      sum = 1.69*x[i];
      for (j = 0; j < n; ++j)
        sum += x[j]*(double)ABS(i - j);
      hx[i] = sum;
    }
}                               /* qphess1 */
```

6.2. Program Data

None, but there is an example data file which contains data and optional parameter values for Example 2 below.

6.3. Program Results

```
e04nfc Example Program Results.

Example 1: default options used.

Parameters to e04nfc
--------------------

Linear constraints............   7    Number of variables..........   8

prob....................  Nag_QP2    start...................  Nag_Cold
ftol....................  1.05e-08   reset_ftol..............         5
rank_tol................  1.11e-14   crash_tol...............  1.00e-02
fcheck..................        50   max_df..................         8
inf_bound...............  1.00e+20   inf_step................  1.00e+20
fmax_iter...............        75   max_iter................        75
```

```
hrows..................          8    machine precision.......  1.11e-16
optim_tol.............. 1.72e-13    min_infeas.............    FALSE
print_level......... Nag_Soln_Iter
outfile................   stdout

Memory allocation:
state..................       Nag
ax.....................       Nag    lambda.................       Nag

Results from e04nfc:
-------------------

  Itn Jdel  Jadd   Step    Ninf  Sinf/Obj    Bnd  Lin  Nart  Nrz  Norm Gz

   0   0     0    0.0e+00   0   0.0000e+00    1    1    6     0   0.00e+00

 Itn 0  -- Feasible point found.
   0   0     0    0.0e+00   0   1.5164e+03    1    1    5     1   9.75e+01
   1   0     8 U  2.8e-01   0   1.7238e+02    2    1    5     0   0.00e+00
   2   1 L  10 L  3.1e-03   0   1.6808e+02    1    2    5     0   0.00e+00
   3   5 A  11 L  1.2e-02   0   1.5718e+02    1    3    4     0   0.00e+00
   4   4 A  12 L  3.2e-02   0   1.3853e+02    1    4    3     0   0.00e+00
   5   3 A  13 L  6.9e-02   0   1.1130e+02    1    5    2     0   0.00e+00
   6   2 A  14 L  1.3e-01   0   7.4123e+01    1    6    1     0   0.00e+00
   7   1 A   1 U  8.4e-01   0  -5.8516e+01    2    6    0     0   0.00e+00
   8  13 L   0    1.0e+00   0  -8.7214e+01    2    5    0     1   0.00e+00
   9   1 U   6 U  2.5e+00   0  -3.1274e+02    2    5    0     1   1.35e+02
  10   0     1 L  1.4e-01   0  -5.6227e+02    3    5    0     0   0.00e+00
  11  14 L   7 U  1.3e-01   0  -6.2149e+02    4    4    0     0   0.00e+00

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1    LL  -1.00000e+00  -1.0000e+00   1.0000e+00    3.045e+02   0.000e+00
 V   2    FR  -2.00000e+00  -2.1000e+00   2.0000e+00    0.000e+00   1.000e-01
 V   3    FR  -3.05000e+00  -3.2000e+00   3.0000e+00    0.000e+00   1.500e-01
 V   4    FR  -4.15000e+00  -4.3000e+00   4.0000e+00    0.000e+00   1.500e-01
 V   5    FR  -5.30000e+00  -5.4000e+00   5.0000e+00    0.000e+00   1.000e-01
 V   6    UL   6.00000e+00  -6.5000e+00   6.0000e+00   -6.100e-01   0.000e+00
 V   7    UL   7.00000e+00  -7.6000e+00   7.0000e+00   -2.442e+01   0.000e+00
 V   8    UL   8.00000e+00  -8.7000e+00   8.0000e+00   -3.423e+01   0.000e+00

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1    LL  -1.00000e+00  -1.0000e+00   None       2.129e+02   1.110e-16
 L   2    LL  -1.05000e+00  -1.0500e+00   None       1.315e+02   0.000e+00
 L   3    LL  -1.10000e+00  -1.1000e+00   None       6.443e+01   0.000e+00
 L   4    LL  -1.15000e+00  -1.1500e+00   None       1.779e+01  -4.441e-16
 L   5    FR   1.13000e+01  -1.2000e+00   None       0.000e+00   1.250e+01
 L   6    FR   1.00000e+00  -1.2500e+00   None       0.000e+00   2.250e+00
 L   7    FR   1.00000e+00  -1.3000e+00   None       0.000e+00   2.300e+00

Exit after 11 iterations.

Optimal QP solution found.

Final QP objective value =  -6.2148782e+02

Re-solve problem with the Hessian defined by function qphess1.

Parameters to e04nfc
--------------------

Linear constraints...........   7    Number of variables..........   8

prob...................   Nag_QP2    start..................  Nag_Cold
ftol................... 1.05e-08    reset_ftol.............         5
rank_tol............... 1.11e-14    crash_tol.............. 1.00e-02
fcheck.................       50    max_df.................         8
```

```
inf_bound..............  1.00e+20    inf_step...............  1.00e+20
fmax_iter..............        75    max_iter...............        75
hrows..................         8    machine precision......  1.11e-16
optim_tol..............  1.72e-13    min_infeas.............     FALSE
print_level......... Nag_Soln_Iter
outfile................    stdout

Memory allocation:
state..................       Nag
ax.....................       Nag    lambda.................       Nag

Results from e04nfc:
-------------------

  Itn Jdel   Jadd    Step    Ninf  Sinf/Obj    Bnd  Lin  Nart  Nrz  Norm Gz

   0   0      0    0.0e+00   3   2.3550e+01    5    0    3     0   1.73e+00
   1    2 U  10 L  4.0e+00   2   1.9600e+01    4    1    3     0   1.41e+00
   2    4 U  12 L  7.8e+00   1   1.1750e+01    3    2    3     0   1.00e+00
   3    6 U  14 L  1.2e+01   0   0.0000e+00    2    3    3     0   0.00e+00

 Itn 3  -- Feasible point found.
   3   0      0    0.0e+00   0   8.6653e+02    2    3    2     1   1.52e+02
   4   0      9 L  1.0e-01   0   4.9824e+01    2    4    2     0   0.00e+00
   5    2 A  11 L  4.5e-01   0  -5.6227e+02    2    5    1     0   0.00e+00
   6    1 A   6 U  6.0e-11   0  -5.6227e+02    3    5    0     0   0.00e+00
   7   14 L   7 U  1.3e-01   0  -6.2149e+02    4    4    0     0   0.00e+00

Final solution:

  Varbl State     Value       Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1     LL  -1.00000e+00  -1.0000e+00   1.0000e+00    3.045e+02   0.000e+00
 V   2     FR  -2.00000e+00  -2.1000e+00   2.0000e+00    0.000e+00   1.000e-01
 V   3     FR  -3.05000e+00  -3.2000e+00   3.0000e+00    0.000e+00   1.500e-01
 V   4     FR  -4.15000e+00  -4.3000e+00   4.0000e+00    0.000e+00   1.500e-01
 V   5     FR  -5.30000e+00  -5.4000e+00   5.0000e+00    0.000e+00   1.000e-01
 V   6     UL   6.00000e+00  -6.5000e+00   6.0000e+00   -6.100e-01   0.000e+00
 V   7     UL   7.00000e+00  -7.6000e+00   7.0000e+00   -2.442e+01   0.000e+00
 V   8     UL   8.00000e+00  -8.7000e+00   8.0000e+00   -3.423e+01   0.000e+00

   LCon  State     Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1     LL  -1.00000e+00  -1.0000e+00   None       2.129e+02   0.000e+00
 L   2     LL  -1.05000e+00  -1.0500e+00   None       1.315e+02   2.220e-16
 L   3     LL  -1.10000e+00  -1.1000e+00   None       6.443e+01  -4.441e-16
 L   4     LL  -1.15000e+00  -1.1500e+00   None       1.779e+01  -4.441e-16
 L   5     FR   1.13000e+01  -1.2000e+00   None       0.000e+00   1.250e+01
 L   6     FR   1.00000e+00  -1.2500e+00   None       0.000e+00   2.250e+00
 L   7     FR   1.00000e+00  -1.3000e+00   None       0.000e+00   2.300e+00

Exit after 7 iterations.

Optimal QP solution found.

Final QP objective value =  -6.2148782e+02
```

7. Further Description

This section gives a detailed description of the algorithm used in nag_opt_qp. This, and possibly the next section, Section 8, may be omitted if the more sophisticated features of the algorithm and software are not currently of interest.

7.1. Overview

nag_opt_qp is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see below). The method is based on that of Gill and Murray (1978) and is described in detail by Gill *et al* (1991). Here we briefly summarize the main features of the method. Where possible, explicit reference is made to the names of variables that are parameters of nag_opt_qp or appear in the printed output. nag_opt_qp has two 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 routines. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function. The feasibility phase does *not* perform the standard simplex method (i.e., it does not necessarily find a vertex), except in the LP case when $m_{lin} \leq n$. Once any iterate is feasible, all subsequent iterates remain feasible.

nag_opt_qp has been designed to be efficient when used to solve a *sequence* of related problems – for example, within a sequential quadratic programming method for nonlinearly constrained optimization. In particular, the user may specify an initial working set (the indices of the constraints believed to be satisfied exactly at the solution); see the discussion of the optional parameter **start** in Section 8.2.

In general, an iterative process is required to solve a quadratic program. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{1}$$

where the *steplength* α is a non-negative scalar, and p is called the *search direction.*

At each point x, a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the tolerance defined by the optional parameter **ftol**; see Section 8.2). The working set is the current prediction of the constraints that hold with equality at a solution of a linearly constrained QP problem. The search direction is constructed so that the constraints in the working set remain *unaltered* for any value of the step length. For a bound constraint in the working set, this property is achieved by setting the corresponding component of the search direction to zero. Thus, the associated variable is *fixed*, and specification of the working set induces a partition of x into *fixed* and *free* variables. During a given iteration, the fixed variables are effectively removed from the problem; since the relevant components of the search direction are zero, the columns of A corresponding to fixed variables may be ignored.

Let m_w denote the number of general constraints in the working set and let n_{fx} denote the number of variables fixed at one of their bounds (m_w and n_{fx} are the quantities `Lin` and `Bnd` in the printed output from nag_opt_qp). Similarly, let n_{fr} ($n_{fr} = n - n_{fx}$) denote the number of free variables. At every iteration, *the variables are re-ordered so that the last n_{fx} variables are fixed*, with all other relevant vectors and matrices ordered accordingly.

7.2. Definition of the Search Direction

Let A_{fr} denote the m_w by n_{fr} sub-matrix of general constraints in the working set corresponding to the free variables, and let p_{fr} denote the search direction with respect to the free variables only. The general constraints in the working set will be unaltered by any move along p if

$$A_{fr} p_{fr} = 0. \tag{2}$$

In order to compute p_{fr}, the TQ factorization of A_{fr} is used:

$$A_{fr} Q_{fr} = (0 \; T), \tag{3}$$

where T is a non-singular m_w by m_w upper triangular matrix (i.e., $t_{ij} = 0$ if $i > j$), and the non-singular n_{fr} by n_{fr} matrix Q_{fr} is the product of orthogonal transformations (see Gill *et al* (1984)). If the columns of Q_{fr} are partitioned so that

$$Q_{fr} = (Z \; Y),$$

where Y is $n_{fr} \times m_w$, then the n_z ($n_z = n_{fr} - m_w$) columns of Z form a basis for the null space of A_{fr}. Let n_r be an integer such that $0 \leq n_r \leq n_z$, and let Z_r denote a matrix whose n_r columns are a subset of the columns of Z. (The integer n_r is the quantity `Nrz` in the printed output from nag_opt_qp. In many cases, Z_r will include *all* the columns of Z.) The direction p_{fr} will satisfy (2) if

$$p_{fr} = Z_r p_r, \tag{4}$$

where p_r is any n_r-vector.

Let Q denote the n by n matrix

$$Q = \begin{pmatrix} Q_{fr} & \\ & I_{fx} \end{pmatrix},$$

where I_{fx} is the identity matrix of order n_{fx}. Let H_q and g_q denote the n by n *transformed Hessian* and the *transformed gradient*

$$H_q = Q^T H Q \quad \text{and} \quad g_q = Q^T(c + Hx)$$

and let the matrix of first n_r rows and columns of H_q be denoted by H_r and the vector of the first n_r elements of g_q be denoted by g_r. The quantities H_r and g_r are known as the *reduced Hessian* and *reduced gradient* of $f(x)$, respectively. Roughly speaking, g_r and H_r describe the first and second derivatives of an *unconstrained* problem for the calculation of p_r.

At each iteration, a triangular factorization of H_r is available. If H_r is positive-definite, $H_r = R^T R$, where R is the upper triangular Cholesky factor of H_r. If H_r is not positive-definite, $H_r = R^T D R$, where $D = \text{diag}(1, 1, \ldots, 1, \mu)$, with $\mu \leq 0$.

The computation is arranged so that the reduced gradient vector is a multiple of e_r, a vector of all zeros except in the last (i.e., n_rth) position. This allows the vector p_r in (4) to be computed from a single back-substitution

$$R p_r = \gamma e_r, \tag{5}$$

where γ is a scalar that depends on whether or not the reduced Hessian is positive-definite at x. In the positive-definite case, $x + p$ is the minimizer of the objective function subject to the constraints (bounds and general) in the working set treated as equalities. If H_r is not positive-definite, p_r satisfies the conditions

$$p_r^T H_r p_r < 0 \quad \text{and} \quad g_r^T p_r \leq 0,$$

which allow the objective function to be reduced by any positive step of the form $x + \alpha p$.

7.3. The Main Iteration

If the reduced gradient is zero, x is a constrained stationary point in the subspace defined by Z. During the feasibility phase, the reduced gradient will usually be zero only at a vertex (although it may be zero at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero reduced gradient implies that x minimizes the quadratic objective when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ_c and λ_b for the general and bound constraints are defined from the equations

$$A_{fr}^T \lambda_c = g_{fr} \quad \text{and} \quad \lambda_b = g_{fx} - A_{fx}^T \lambda_c. \tag{6}$$

Given a positive constant δ of the order of the ***machine precision***, a Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \delta$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\delta$ when the associated constraint is at its *lower bound*. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint (with index `Jdel`; see Section 8.3) from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is non-zero, there is no feasible point, and the user can force nag_opt_qp to continue until the minimum value of the sum of infeasibilities has been found (see the discussion of the optional parameter **min_infeas** in Section 8.2). At this point, the Lagrange multiplier λ_j corresponding to an inequality constraint in the working set will be such that $-(1+\delta) \leq \lambda_j \leq \delta$ when the associated constraint is at its *upper bound*, and $-\delta \leq \lambda_j \leq 1+\delta$ when the associated constraint is at its *lower bound*. Lagrange multipliers for equality constraints will satisfy $||\lambda_j|| \leq 1+\delta$.

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the non-zero elements of the search direction p are given by $Z_r p_r$ (see (5)). The choice of step length is influenced by the need to maintain feasibility with respect to the satisfied constraints. If H_r is positive-definite and $x+p$ is feasible, α will be taken as unity. In this case, the reduced gradient at $\bar{x}$ will be zero, and Lagrange multipliers are computed. Otherwise, α is set to α_m, the step to the 'nearest' constraint (with index `Jadd`; see Section 8.3), which is added to the working set at the next iteration.

Each change in the working set leads to a simple change to A_{fr}: if the status of a general constraint changes, a *row* of A_{fr} is altered; if a bound constraint enters or leaves the working set, a *column* of A_{fr} changes. Explicit representations are recurred of the matrices T, Q_{fr} and R; and of vectors $Q^T g$, and $Q^T c$. The triangular factor R associated with the reduced Hessian is only updated during the optimality phase.

One of the most important features of nag_opt_qp is its control of the conditioning of the working set, whose nearness to linear dependence is estimated by the ratio of the largest to smallest diagonal elements of the TQ factor T (the printed value `Cond T`; see Section 8.3). In constructing the initial working set, constraints are excluded that would result in a large value of `Cond T`.

nag_opt_qp includes a rigorous procedure that prevents the possibility of cycling at a point where the active constraints are nearly linearly dependent (see Gill *et al* (1989)). The main feature of the anti-cycling procedure is that the feasibility tolerance is increased slightly at the start of every iteration. This not only allows a positive step to be taken at every iteration, but also provides, whenever possible, a *choice* of constraints to be added to the working set. Let α_m denote the maximum step at which $x+\alpha_m p$ does not violate any constraint by more than its feasibility tolerance. All constraints at a distance α ($\alpha \leq \alpha_m$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the largest angle with the search direction is added to the working set.

7.4. Choosing the Initial Working Set

At the start of the optimality phase, a positive-definite H_r can be defined if enough constraints are included in the initial working set. (The matrix with no rows and columns is positive-definite by definition, corresponding to the case when A_{fr} contains n_{fr} constraints.) The idea is to include as many general constraints as necessary to ensure that the reduced Hessian is positive-definite.

Let H_z denote the matrix of the first n_z rows and columns of the matrix $H_q = Q^T HQ$ at the beginning of the optimality phase. A partial Cholesky factorization is used to find an upper triangular matrix R that is the factor of the largest positive-definite leading sub-matrix of H_z. The use of interchanges during the factorization of H_z tends to maximize the dimension of R. (The condition of R may be controlled using the optional parameter **rank_tol**; see Section 8.2.) Let Z_r denote the columns of Z corresponding to R, and let Z be partitioned as $Z = (Z_r Z_a)$. A working set for which Z_r defines the null space can be obtained by including *the rows* of Z_a^T as 'artificial constraints'. Minimization of the objective function then proceeds within the subspace defined by Z_r, as described in Section 7.2.

The artificially augmented working set is given by

$$\bar{A}_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix}, \tag{7}$$

so that p_{fr} will satisfy $A_{fr} p_{fr} = 0$ *and* ${Z_a}^T p_{fr} = 0$. By definition of the TQ factorization, $\bar{A}_{fr}$ *automatically* satisfies the following:

$$\bar{A}_{fr} Q_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix} Q_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix} (Z_r \; Z_a \; Y) = (0 \; \bar{T}),$$

where

$$\bar{T} = \begin{pmatrix} I & 0 \\ 0 & T \end{pmatrix},$$

and hence the TQ factorization of (7) is available trivially from T and Q_{fr} without additional expense.

The matrix Z_a is not kept fixed, since its role is purely to define an appropriate null space; the TQ factorization can therefore be updated in the normal fashion as the iterations proceed. No work is required to 'delete' the artificial constraints associated with Z_a when ${Z_r}^T g_{fr} = 0$, since this simply involves repartitioning Q_{fr}. The 'artificial' multiplier vector associated with the rows of ${Z_a}^T$ is equal to ${Z_a}^T g_{fr}$, and the multipliers corresponding to the rows of the 'true' working set are the multipliers that would be obtained if the artificial constraints were not present. If an artificial constraint is 'deleted' from the working set, an `A` appears alongside the entry in the `Jdel` column of the printed output (see Section 8.3).

The number of columns in Z_a and Z_r, the Euclidean norm of ${Z_r}^T g_{fr}$, and the condition estimator of R appear in the printed output as `Nart`, `Nrz`, `Norm Gz` and `Cond Rz` (see Section 8.3).

Under some circumstances, a different type of artificial constraint is used when solving a linear program. Although the algorithm of nag_opt_qp does not usually perform simplex steps (in the traditional sense), there is one exception: a linear program with fewer general constraints than variables (i.e., $m_{lin} \leq n$). (Use of the simplex method in this situation leads to savings in storage.) At the starting point, the 'natural' working set (the set of constraints exactly or nearly satisfied at the starting point) is augmented with a suitable number of 'temporary' bounds, each of which has the effect of temporarily fixing a variable at its current value. In subsequent iterations, a temporary bound is treated as a standard constraint until it is deleted from the working set, in which case it is never added again. If a temporary bound is 'deleted' from the working set, an `F` (for 'Fixed') appears alongside the entry in the `Jdel` column of the printed output (see Section 8.3).

8. Optional Parameters

A number of optional input and output parameters to nag_opt_qp are available through the structure argument **options**, type Nag_E04_Opt. A parameter may be selected by assigning an appropriate value to the relevant structure member; those parameters not selected will be assigned default values. If no use is to be made of any of the optional parameters the user should use the NAG defined null pointer, `E04_DEFAULT`, in place of **options** when calling nag_opt_qp; the default settings will then be used for all parameters.

Before assigning values to **options** directly the structure **must** be initialized by a call to the function nag_opt_init (e04xxc). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a text file using the function nag_opt_read (e04xyc) in which case initialization of the **options** structure will be performed automatically if not already done. Any subsequent direct assignment to the **options** structure must **not** be preceded by initialization.

If assignment of functions and memory to pointers in the **options** structure is required, this must be done directly in the calling program; they cannot be assigned using nag_opt_read (e04xyc).

8.1. Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag_opt_qp together with their default values where relevant. The number ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

`Nag_ProblemType prob`	**Nag_QP2**
`Nag_Start start`	**Nag_Cold**
`Boolean list`	**TRUE**
`Nag_PrintType print_level`	**Nag_Soln_Iter**
`char outfile[80]`	`stdout`
`void (*print_fun)()`	NULL
`Integer fmax_iter`	max(50,5(**n**+**nclin**))
`Integer max_iter`	max(50,5(**n**+**nclin**))
`Boolean min_infeas`	**FALSE**
`double crash_tol`	0.01
`double ftol`	$\sqrt{\epsilon}$
`double optim_tol`	$\epsilon^{0.8}$
`Integer reset_ftol`	10000
`Integer fcheck`	50
`double inf_bound`	10^{20}
`double inf_step`	max(**inf_bound**,10^{20})
`Integer hrows`	**n**
`Integer max_df`	**n**
`double rank_tol`	100ϵ
`Integer *state`	size **n**+**nclin**
`double *ax`	size **nclin**
`double *lambda`	size **n**+**nclin**
`Integer iter`	
`Integer nf`	

8.2. Description of Optional Parameters

prob – Nag_ProblemType Default = **Nag_QP2**

Input: specifies the type of objective function to be minimized during the optimality phase. The following are the six possible values of **prob** and the size of the arrays **h** and **cvec** that are required to define the objective function:

Nag_FP **h** and **cvec** not accessed;

Nag_LP **h** not accessed, **cvec**[**n**] required;

Nag_QP1 **h**[**n**∗**tdh**] symmetric, **cvec** not referenced;

Nag_QP2 **h**[**n**∗**tdh**] symmetric, **cvec**[**n**] required;

Nag_QP3 **h**[**n**∗**tdh**] upper trapezoidal, **cvec** not referenced;

Nag_QP4 **h**[**n**∗**tdh**] upper trapezoidal, **cvec**[**n**] required.

If $H = 0$, i.e., the objective function is purely linear, the efficiency of nag_opt_qp may be increased by specifying **prob** as **Nag_LP**.
Constraint: **options.prob** = **Nag_FP** or **Nag_LP** or **Nag_QP1** or **Nag_QP2** or **Nag_QP3** or **Nag_QP4**.

start – Nag_Start Default = **Nag_Cold**

Input: specifies how the initial working set is chosen. With **options.start** = **Nag_Cold**, nag_opt_qp chooses the initial working set based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or 'nearly' satisfy their bounds (to within **crash_tol**; see below).

With **options.start** = **Nag_Warm**, the user must provide a valid definition of every element of the array pointer **options.state** (see below for the definition of this member of **options**). nag_opt_qp will override the users' specification of **state** if necessary, so that a poor choice of the working set will not cause a fatal error. **Nag_Warm** will be advantageous if a good estimate of the initial working set is available – for example, when nag_opt_qp is called repeatedly to solve related problems.
Constraint: **options.start** = **Nag_Cold** or **Nag_Warm**.

list – Boolean Default = **TRUE**

Input: if **options.list** = **TRUE** the parameter settings in the call to nag_opt_qp will be printed.

print_level – Nag_PrintType Default = **Nag_Soln_Iter**

Input: the level of results printout produced by nag_opt_qp. The following values are available.

Nag_NoPrint	No output.
Nag_Soln	The final solution.
Nag_Iter	One line of output for each iteration.
Nag_Iter_Long	A longer line of output for each iteration with more information (line exceeds 80 characters).
Nag_Soln_Iter	The final solution and one line of output for each iteration.
Nag_Soln_Iter_Long	The final solution and one long line of output for each iteration (line exceeds 80 characters).
Nag_Soln_Iter_Const	As **Nag_Soln_Iter_Long** with the Lagrange multipliers, the variables x, the constraint values Ax and the constraint status also printed at each iteration.
Nag_Soln_Iter_Full	As **Nag_Soln_Iter_Const** with the diagonal elements of the upper triangular matrix T associated with the TQ factorization (3) of the working set, and the diagonal elements of the upper triangular matrix R printed at each iteration.

Details of each level of results printout are described in Section 8.3.
Constraint: **options.print_level** = **Nag_NoPrint** or **Nag_Soln** or **Nag_Iter** or **Nag_Soln_Iter** or **Nag_Iter_Long** or **Nag_Soln_Iter_Long** or **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full**.

outfile – char[80] Default = `stdout`

Input: the name of the file to which results should be printed. If **options.outfile**[0] = '\0' then the `stdout` stream is used.

print_fun – pointer to function Default = NULL

Input: printing function defined by the user; the prototype of **print_fun** is

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

See Section 8.3.1 below for further details.

fmax_iter – Integer Default = max(50,5(**n**+**nclin**))
max_iter – Integer Default = max(50,5(**n**+**nclin**))

Input: **fmax_iter** specifies the maximum number of iterations allowed in the feasibility phase. **max_iter** specifies the maximum number of iterations permitted in the optimality phase.

If the user wishes to check that a call to nag_opt_qp is correct before attempting to solve the problem in full then **fmax_iter** may be set to 0. No iterations will then be performed but the initialization stages prior to the first iteration will be processed and a listing of parameter settings output, if **options.list** = **TRUE** (the default setting).
Constraint: **options.fmax_iter** ≥ 0 and **options.max_iter** ≥ 0.

min_infeas – Boolean Default = **FALSE**

Input: **min_infeas** specifies whether nag_opt_qp should minimize the sum of infeasibilities if no feasible point exists for the constraints. If **min_infeas** = **FALSE** then nag_opt_qp will terminate as soon as it is evident that the problem is infeasible, in which case the final point will generally not be the point at which the sum of infeasibilities is minimized. If **min_infeas** = **TRUE**, nag_opt_qp will continue until the sum of infeasibilities is minimized.

crash_tol – double Default = 0.01

Input: **crash_tol** is used in conjunction with the optional parameter **start** when **start** has the default setting, i.e., **options.start** = **Nag_Cold**, nag_opt_qp selects an initial working set. The initial working set will include bounds or general inequality constraints that lie within **crash_tol** of their bounds. In particular, a constraint of the form ${a_j}^T x \geq l$ will be included in the initial working set if $|{a_j}^T x - l| \leq$ **crash_tol** $\times$ $(1 + |l|)$.
Constraint: $0.0 \leq$ **options.crash_tol** ≤ 1.0.

ftol – double Default = $\sqrt{\epsilon}$

Input: **ftol** defines the maximum acceptable *absolute* violation in each constraint at a 'feasible' point. For example, if the variables and the coefficients in the general constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify **ftol** as 10^{-6}.

nag_opt_qp attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, **options.min_infeas** (see above) can be used to find the minimum value of the sum. Let `Sinf` be the corresponding sum of infeasibilities. If `Sinf` is quite small, it may be appropriate to raise **ftol** by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

Note that a 'feasible solution' is a solution that satisfies the current constraints to within the tolerance **ftol**.
Constraint: **options.ftol** > 0.0.

optim_tol – double Default = $\epsilon^{0.8}$

Input: **options.optim_tol** defines the tolerance used to determine whether the bounds and generated constraints have the correct sign for the solution to be judged optimal.

reset_ftol – Integer Default = 5

Input: this option is part of an anti-cycling procedure designed to guarantee 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 parameter **ftol** is δ. Over a period of **reset_ftol** iterations, the feasibility tolerance actually used by nag_opt_qp increases from 0.5δ to δ (in steps of 0.5δ/**reset_ftol**).

At certain stages the following 'resetting procedure' is used to remove constraint infeasibilities. First, all variables whose upper or lower bounds are in the working set are moved exactly onto their bounds. A count is kept of the number of nontrivial adjustments made. If the count is positive, iterative refinement is used to give variables that satisfy the working set to (essentially) ***machine precision***. Finally, the current feasibility tolerance is reinitialized to 0.5δ.

If a problem requires more than **reset_ftol** iterations, the resetting procedure is invoked and a new cycle of **reset_ftol** iterations is started with **reset_ftol** incremented by 10. (The decision to resume the feasibility phase or optimality phase is based on comparing any constraint infeasibilities with δ.)

The resetting procedure is also invoked when nag_opt_qp reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any nontrivial adjustments are made, iterations are continued.
Constraint: $0 <$ **options.reset_ftol** < 10000000.

fcheck – Integer Default = 50

Input: every **fcheck** iterations, a numerical test is made to see if the current solution x satisfies the constraints in the working set. If the largest residual of the constraints in the working set is judged to be too large, the current working set is re-factorized and the variables are recomputed to satisfy the constraints more accurately.
Constraint: **options.fcheck** ≥ 1.

inf_bound – double Default = 10^{20}

Input: **inf_bound** defines the 'infinite' bound in the definition of the problem constraints. Any upper bound greater than or equal to **inf_bound** will be regarded as plus infinity (and similarly for a lower bound less than or equal to −**inf_bound**).
Constraint: **options.inf_bound** > 0.0.

inf_step – double Default = max(**inf_bound**,10^{20})

Input: **inf_step** 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 **inf_step**, the objective function is considered to be unbounded below in the feasible region.
Constraint: **options.inf_step** > 0.0.

hrows – Integer Default = **n**

Input: specifies m, the number of rows of the quadratic term H of the QP objective function. The default value of **hrows** is n, the number of variables of the problem, except that if the problem is specified as type **Nag_FP** or **Nag_LP**, the default value of **hrows** is zero.

If the problem is of type QP, **hrows** will usually be n, the number of variables. However, a value of **hrows** less than n is appropriate for **Nag_QP3** or **Nag_QP4** if H is an upper trapezoidal matrix with m rows. Similarly, **hrows** may be used to define the dimension of a leading block of non-zeros in the Hessian matrices of **Nag_QP1** or **Nag_QP2**, in which case the last $n - m$ rows and columns of H are assumed to be zero.
Constraint: $0 \leq$ **options.hrows** $\leq$ **n**.

max_df – Integer Default = **n**

Input: places a limit on the storage allocated for the triangular factor R of the reduced Hessian H_r. Ideally, **max_df** should be set slightly larger than the value of n_r expected at the solution. It need not be larger than $m_n + 1$, where m_n is the number of variables that appear nonlinearly in the quadratic objective function. For many problems it can be much smaller than m_n.

For quadratic problems, a minimizer may lie on any number of constraints, so that n_r may vary between 1 and n. The default value is therefore normally **n** but if the optional parameter **hrows** is specified then the default value of **max_df** is set to the value in **hrows**.
Constraint: $1 \leq$ **options.max_df** $\leq$ **n**.

rank_tol – double Default = 100ϵ

Input: **rank_tol** enables the user to control the condition number of the triangular factor R (see Section 7). If ρ_i denotes the function $\rho_i = \max\{|R_{11}|, |R_{22}|, \ldots, |R_{ii}|\}$, the dimension of R is defined to be smallest index i such that $|R_{i+1,i+1}| \leq$ **rank_tol** $\times$ $|\rho_{i+1}|$.
Constraint: $0.0 \leq$ **options.rank_tol** < 1.0.

state – Integer * Default memory = **n+nclin**

Input: **state** need not be set if the default option of **options.start** = **Nag_Cold** is used as **n+nclin** values of memory will be automatically allocated by nag_opt_qp.

If the option **start** = **Nag_Warm** has been chosen, **state** must point to a minimum of **n+nclin** elements of memory. This memory will already be available if the **options** structure has been used in a previous call to nag_opt_qp from the calling program, using the same values of **n** and **nclin** and **start** = **Nag_Cold**. If a previous call has not been made sufficient memory must be allocated to **state** by the user.

When a warm start is chosen **state** should specify the desired status of the constraints at the start of the feasibility phase. More precisely, the first n elements of **state** refer to the upper and lower bounds on the variables, and the next m_{lin} elements refer to the general linear constraints (if any). Possible values for **state**[j] are as follows:

state[j]	**Meaning**
0	The corresponding constraint should *not* be in the initial working set.
1	The constraint should be in the initial working set at its lower bound.
2	The constraint should be in the initial working set at its upper bound.
3	The constraint should be in the initial working set as an equality. This value should only be specified if **bl**[j] = **bu**[j]. The values 1,2 or 3 all have the same effect when **bl**[j] = **bu**[j].

The values −2, −1 and 4 are also acceptable but will be reset to zero by the function. In particular, if nag_opt_qp has been called previously with the same values of **n** and **nclin**, **state** already contains satisfactory information. (See also the description of the optional parameter **start**.) The function also adjusts (if necessary) the values supplied in **x** to be consistent with the values supplied in **state**.

Output: if nag_opt_qp exits with a value of **fail.code** = **NE_NOERROR**, **NW_DEAD_POINT**, **NW_SOLN_NOT_UNIQUE** or **NW_NOT_FEASIBLE**, the values in **state** indicate the status of the constraints in the working set at the solution. Otherwise, **state** indicates the composition of the working set at the final iterate. The significance of each possible value of **state**[j] is as follows:

state[j]	**Meaning**
−2	The constraint violates its lower bound by more than the feasibility tolerance.
−1	The constraint violates its upper bound by more than the feasibility tolerance.
0	The constraint is satisfied to within the feasibility tolerance, but is not in the working set.
1	This inequality constraint is included in the working set at its lower bound.
2	This inequality constraint is included in the working set at its upper bound.
3	This constraint is included in the working set as an equality. This value of **state** can occur only when **bl**[j] = **bu**[j].
4	This corresponds to optimality being declared with **x**[j] being temporarily fixed at its current value. This value of **state** can only occur when **fail.code** = **NW_DEAD_POINT** or **NW_SOLN_NOT_UNIQUE**.

ax – double * Default memory = **nclin**

Input: **nclin** values of memory will be automatically allocated by nag_opt_qp and this is the recommended method of use of **options.ax**. However a user may supply memory from the calling program.
Output: if **nclin** > 0, **ax** points to the final values of the linear constraints Ax.

lambda – double * Default memory = **n**+**nclin**

Input: **n**+**nclin** values of memory will be automatically allocated by nag_opt_qp and this is the recommended method of use of **options.lambda**. However a user may supply memory from the calling program.
Output: the values of the Lagrange multipliers for each constraint with respect to the current working set. The first n elements contain the multipliers for the bound constraints on the variables, and the next m_{lin} elements contain the multipliers for the general linear constraints (if any). If **state**[j] = 0 (i.e., constraint j is not in the working set), **lambda**[j] is zero. If x is optimal, **lambda**[j] should be non-negative if **state**[j] = 1, non-positive if **state**[j] = 2 and zero if **state**[j] = 4.

iter – Integer

Output: the total number of iterations performed in the feasibility phase and (if appropriate) the optimality phase.

nf – Integer

Output: the number of times the product Hx has been calculated (i.e., number of calls of **qphess**).

8.3. Description of Printed Output

The level of printed output can be controlled by the user with the structure members **options.list** and **options.print_level** (see Section 8.2). If **list** = **TRUE** then the parameter values to nag_opt_qp are listed, whereas the printout of results is governed by the value of **print_level**. The default of **print_level** = **Nag_Soln_Iter** provides a single line of output at each iteration and the final result. This section describes all of the possible levels of results printout available from nag_opt_qp.

The convention for numbering the constraints in the iteration results is that indices 1 to n refer to the bounds on the variables, and indices $n + 1$ to $n + m_{lin}$ refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

When **print_level** = **Nag_Iter** or **Nag_Soln_Iter** the following line of output is produced at every iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn` the iteration count.

`Jdel` the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. During the optimality phase, the step can be greater than 1.0 only if the reduced Hessian is not positive-definite.

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

`Sinf/Obj` the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If x is feasible, `Obj` 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. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Bnd` the number of simple bound constraints in the current working set.

`Lin` the number of general linear constraints in the current working set.

`Nart` the number of artificial constraints in the working set, i.e., the number of columns of Z_a (see Section 7). At the start of the optimality phase, `Nart` provides an estimate of the number of nonpositive eigenvalues in the reduced Hessian.

`Nrz` the number of columns of Z_r (see Section 7). `Nrz` is the dimension of the subspace in which the objective function is currently being minimized. The value of `Nrz` is the number of variables minus the number of constraints in the working set; i.e., $\texttt{Nrz} = n - (\texttt{Bnd} + \texttt{Lin} + \texttt{Nart})$.

The value of n_z, the number of columns of Z (see Section 7) can be calculated as $n_z = n - (\texttt{Bnd} + \texttt{Lin})$. A zero value of n_z implies that x lies at a vertex of the feasible region.

`Norm Gz` $\|Z_r{}^T g_{fr}\|$, the Euclidean norm of the reduced gradient with respect to Z_r. During the optimality phase, this norm will be approximately zero after a unit step.

If **print_level** = **Nag_Iter_Long**, **Nag_Soln_Iter_Long**, **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** the line of printout is extended to give the following information. (Note this longer line extends over more than 80 characters.)

`NOpt` the number of non-optimal Lagrange multipliers at the current point. `NOpt` is not printed if the current x is infeasible or no multipliers have been calculated. At a minimizer, `NOpt` will be zero.

`Min LM` the value of the Lagrange multiplier associated with the deleted constraint. If `Min LM` is negative, a lower bound constraint has been deleted; if `Min LM` is positive, an upper bound constraint has been deleted. If no multipliers are calculated during a given iteration, `Min LM` will be zero.

`Cond T` a lower bound on the condition number of the working set.

`Cond Rz` a lower bound on the condition number of the triangular factor R (the Cholesky factor of the current reduced Hessian). If the problem is specified to be of type **Nag_LP**, `Cond Rz` is not printed.

`Rzz` the last diagonal element μ of the matrix D associated with the $R^T DR$ factorization of the reduced Hessian H_r (see Section 7.2). `Rzz` is only printed if H_r is not positive-definite (in which case $\mu \neq 1$). If the printed value of `Rzz` is small in absolute value, then H_r is approximately singular. A negative value of `Rzz` implies that the objective function has negative curvature on the current working set.

When **options.print_level** = **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** more detailed results are given at each iteration. For the setting **Nag_Soln_Iter_Const** additional values output are:

`Value of x` the value of x currently held in **x**.

`State` the current value of **options.state** associated with x.

`Value of Ax` the value of Ax currently held in **options.ax**.

`State` the current value of **options.state** associated with Ax.

Also printed are the Lagrange Multipliers for the bound constraints, linear constraints and artificial constraints.

If **print_level** = **Nag_Soln_Iter_Full** then the diagonal of T and Z_r are also output at each iteration.

When **print_level** = **Nag_Soln**, **Nag_Soln_Iter**, **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** the final printout from nag_opt_qp includes a listing of the status of every variable and constraint. The following describes the printout for each variable.

`Varbl` gives the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State` gives the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the feasibility tolerance, `State` will be `++` or `--` respectively.

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

`Lower bound` is the lower bound specified for the variable.
(`None` indicates that **bl**$[j-1] \leq -$**inf_bound**.)

`Upper bound` is the upper bound specified for the variable. (`None` indicates that **bu**[$j-1$] $\geq$ **inf_bound**.)

`Lagr mult` is the value of the Lagrange multiplier for the associated bound constraint. This will be zero if `State` is `FR`. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.

`Residual` is the difference between the variable `Value` and the nearer of its bounds **bl**[$j-1$] and **bu**[$j-1$].

The meaning of the printout for general constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', and with the following change in the heading:

`LCon` is the name (`L`) and index j, for $j = 1, 2, \ldots, m_{lin}$ of the constraint.

8.3.1. Output of results via a user defined printing function

The user may also specify their own print function for output of iteration results and the final solution by use of the **options.print_fun** function pointer, which has prototype

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

The rest of this section can be skipped by a user who only wishes to use the default printing facilities.

When a user defined function is assigned to **options.print_fun** this will be called in preference to the internal print function of nag_opt_qp. Calls to the user defined function are again controlled by means of the **options.print_level** member. Information is provided through **st** and **comm**, the two structure arguments to **print_fun**.

If **comm->it_prt** = **TRUE** then the results from the last iteration of nag_opt_qp are set in the following members of **st**:

first – Boolean
  TRUE on the first call to **print_fun**.

iter – Integer
  the number of iterations performed.

n – Integer
  the number of variables.

nclin – Integer
  the number of linear constraints.

jdel – Integer
  index of constraint deleted.

jadd – Integer
  index of constraint added.

step – double
  the step taken along the current search direction.

ninf – Integer
  the number of infeasibilities.

f – double
  the value of the current objective function.

bnd – Integer
  number of bound constraints in the working set.

lin – Integer
  number of general linear constraints in the working set.

nart – Integer
  number of artificial constraints in the working set.

nrz – Integer
  number of columns of Z_r.

norm_gz – double
Euclidean norm of the reduced gradient, $\|{Z_r}^T g_{fr}\|$.

nopt – Integer
number of non-optimal Lagrange multipliers.

min_lm – double
value of the Lagrange multiplier associated with the deleted constraint.

condt – double
a lower bound on the condition number of the working set.

x – double *
x points to the **n** memory locations holding the current point x.

ax – double *
ax points to the **nclin** memory locations holding the current values Ax.

state – Integer *
state points to the **n**+**nclin** memory locations holding the status of the variables and general linear constraints. See Section 8.2 for a description of the possible status values.

t – double *
the upper triangular matrix T with **st->lin** columns. Matrix element i,j is held in **st->t**[$(i-1)*$**st->tdt**$+j-1$].

tdt – Integer
the trailing dimension for **st->t**.

If **st->rset** = **TRUE** then the problem is QP, nag_opt_qp is executing the optimality phase and the following members of **st** are also set:

r – double *
the upper triangular matrix R with **st->nrz** columns. Matrix element i,j is held in **st->r**[$(i-1)*$**st->tdr**$+j-1$].

tdr – Integer
the trailing dimension for **st->r**.

condr – double
a lower bound on the condition number of the triangular factor R.

rzz – double
last diagonal element μ of the matrix D.

If **comm->new_lm** = **TRUE** then the Lagrange multipliers have been updated and the following members of **st** are set:

kx – Integer *
Indices of the bound constraints with associated multipliers.
Value of **st->kx**[i] is the index of the constraint with multiplier **st->lambda**[i] for $i = 0, 1, \ldots,$**st->bnd**-1.

kactive – Integer *
Indices of the linear constraints with associated multipliers.
Value of **st->kactive**[i] is the index of the constraint with multiplier **st->lambda**[**st->bnd**$+ i$] for $i = 0, 1, \ldots,$**st->lin**-1.

lambda – double *
the multipliers for the constraints in the working set. **lambda**[i] for $i = 0, 1, \ldots,$**st->bnd**-1 hold the multipliers for the bound constraints while the multipliers for the linear constraints are held at indices $i =$ **st->bnd**,...,**st->bnd**+**st->lin**-1.

gq – double *
st->gq[i] for $i = 0, 1, \ldots,$**st->nart**-1 hold the multipliers for the artificial constraints.

The following members of **st** are also relevant and apply when **comm->it_prt** or **comm->new_lm** is **TRUE**.

refactor – Boolean
TRUE if iterative refinement performed. See Section 8.2 and optional parameter **reset_ftol**.

jmax – Integer
if **st->refactor** = **TRUE** then **st->jmax** holds the index of the constraint with the maximum violation.

errmax – double
if **st->refactor** = **TRUE** then **st->errmax** holds the value of the maximum violation.

moved – Boolean
TRUE if some variables have been moved to their bounds. See the optional parameter **reset_ftol**.

nmoved – Integer
if **st->moved** = **TRUE** then **st->nmoved** holds the number of variables which were moved to their bounds.

rowerr – Boolean
TRUE if some constraints are not satisfied to within **options.ftol**.

feasible – Boolean
TRUE when a feasible point has been found.

If **comm->sol_prt** = **TRUE** then the final result from nag_opt_qp is available and the following members of **st** are set:

iter – Integer
the number of iterations performed.

n – Integer
the number of variables.

nclin – Integer
the number of linear constraints.

x – double *
x points to the **n** memory locations holding the final point x.

f – double
the final objective function value or, if x is not feasible, the sum of infeasibilities. If the problem is of type **Nag_FP** and x is feasible then **f** is set to zero.

ax – double *
ax points to the **nclin** memory locations holding the final values Ax.

state – Integer *
state points to the **n**+**nclin** memory locations holding the final status of the variables and general linear constraints. See Section 8.2 for a description of the possible status values.

lambda – double *
lambda points to the **n**+**nclin** final values of the Lagrange multipliers.

bl – double *
bl points to the **n**+**nclin** lower bound values.

bu – double *
bu points to the **n**+**nclin** upper bound values.

endstate – Nag_EndState
the state of termination of nag_opt_qp. Possible values of **endstate** and their correspondence to the exit value of **fail.code** are:

Value of **endstate**	Value of **fail.code**
Nag_Feasible and **Nag_Optimal**	**NE_NOERROR**
Nag_Deadpoint and **Nag_Weakmin**	If the problem is QP **NW_DEADPOINT** otherwise **NW_SOLN_NOT_UNIQUE**
Nag_Unbounded	**NE_UNBOUNDED**
Nag_Infeasible	**NW_NOT_FEASIBLE**
Nag_Too_Many_Iter	**NW_TOO_MANY_ITER**
Nag_Hess_Too_Big	**NE_HESS_TOO_BIG**

The relevant members of the structure **comm** are:

it_prt – Boolean
will be **TRUE** when the print function is called with the result of the current iteration.

sol_prt – Boolean
will be **TRUE** when the print function is called with the final result.

new_lm – Boolean
will be **TRUE** when the Lagrange multipliers have been updated.

user – double *
iuser – Integer *
p – Pointer
pointers for communication of user information. If used they must be allocated memory by the user either before entry to nag_opt_qp or during a call to **qphess** or **print_fun**. The type Pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

9. Error Indications and Warnings

NE_USER_STOP
User requested termination, user flag value = $\langle value\rangle$.

This exit occurs if the user sets **comm->flag** to a negative value in **qphess**. If **fail** is supplied the value of **fail.errnum** will be the same as the user's setting of **comm->flag**.

NE_INT_ARG_LT
On entry, **n** must not be less than 1: **n** = $\langle value\rangle$.
On entry, **nclin** must not be less than 0: **nclin** = $\langle value\rangle$.

NE_2_INT_ARG_LT
On entry, **tda** = $\langle value\rangle$ while **n** = $\langle value\rangle$. These parameters must satisfy **tda** $\geq$ **n**.
On entry, **tdh** = $\langle value\rangle$ while **n** = $\langle value\rangle$. These parameters must satisfy **tdh** $\geq$ **n**.
On entry, **tdh** = $\langle value\rangle$ while **options.hrows** = $\langle value\rangle$. These parameters must satisfy **tdh** $\geq$ **hrows**.

NE_OPT_NOT_INIT
Options structure not initialized.

NE_BAD_PARAM
On entry parameter **options.print_level** had an illegal value.
On entry parameter **options.prob** had an illegal value.
On entry parameter **options.start** had an illegal value.

NE_INVALID_INT_RANGE_1
Value $\langle value\rangle$ given to **options.hrows** not valid. Correct range is **n** $\geq$ **hrows** ≥ 0.
Value $\langle value\rangle$ given to **options.max_df** not valid. Correct range is **n** $\geq$ **max_df** ≥ 1.
Value $\langle value\rangle$ given to **options.max_iter** not valid. Correct range is **max_iter** ≥ 0.
Value $\langle value\rangle$ given to **options.fmax_iter** not valid. Correct range is **fmax_iter** ≥ 0.
Value $\langle value\rangle$ given to **options.fcheck** not valid. Correct range is **fcheck** ≥ 1.

NE_INVALID_INT_RANGE_2
Value $\langle value\rangle$given to **options.reset_ftol** not valid. Correct range is $0 <$ **reset_ftol** < 10000000.

NE_INVALID_REAL_RANGE_FF
Value $\langle value\rangle$ given to **options.crash_tol** not valid. Correct range is $0.0 \leq$ **crash_tol** ≤ 1.0.
Value $\langle value\rangle$ given to **options.rank_tol** not valid. Correct range is $0.0 \leq$ **rank_tol** < 1.0.

NE_INVALID_REAL_RANGE_F

Value ⟨*value*⟩ given to **options.ftol** not valid. Correct range is **ftol** > 0.0.
Value ⟨*value*⟩ given to **options.inf_bound** not valid. Correct range is **inf_bound** > 0.0.
Value ⟨*value*⟩ given to **options.inf_step** not valid. Correct range is **inf_step** > 0.0.

NE_CVEC_NULL

options.prob = ⟨*value*⟩ but argument **cvec** = NULL.

NE_H_NULL

options.prob = ⟨*value*⟩, **qphess** is NULL but argument **h** is also NULL. If the default function for **qphess** is to be used for this problem then an array must be supplied in parameter **h**.

NE_WARM_START

options.start = **Nag_Warm** but pointer **options.state** = NULL.

NE_BOUND

The lower bound for variable ⟨*value*⟩ (array element **bl**[⟨*value*⟩]) is greater than the upper bound.

NE_BOUND_LCON

The lower bound for linear constraint ⟨*value*⟩ (array element **bl**[⟨*value*⟩]) is greater than the upper bound.

NE_STATE_VAL

options.state[⟨*value*⟩] is out of range. **state**[⟨*value*⟩] = ⟨*value*⟩.

NE_ALLOC_FAIL

Memory allocation failed.

If one of the above exits occurs, no values will have been assigned to **objf**, or to **options.ax** and **options.lambda**. **x** and **options.state** will be unchanged.

NW_DEAD_POINT

Iterations terminated at a dead point (check the optimality conditions).

The necessary conditions for optimality have been satisfied but the sufficient conditions are not. (The reduced gradient is negligible, the Lagrange multipliers are optimal, but H_r is singular or there are some very small multipliers.) If H is not positive-definite, x is not necessarily a local solution of the problem and verification of optimality requires further information.

NW_SOLN_NOT_UNIQUE

Optimal solution is not unique.

The necessary conditions for optimality have been satisfied but the sufficient conditions are not. (The reduced gradient is negligible, the Lagrange multipliers are optimal, but H_r is singular or there are some very small multipliers.) If H is positive semi-definite, x gives the global minimum value of the objective function, but the final x is not unique.

NE_UNBOUNDED

Solution appears to be unbounded.

This value of **fail.code** implies that a step as large as **options.inf_step** would have to be taken in order to continue the algorithm. This situation can occur only when H is not positive-definite and at least one variable has no upper or lower bound.

NW_NOT_FEASIBLE

No feasible point was found for the linear constraints.

It was not possible to satisfy all the constraints to within the feasibility tolerance. In this case, the constraint violations at the final x will reveal a value of the tolerance for which a feasible point will exist – for example, if the feasibility tolerance for each violated constraint exceeds its `Residual` at the final point. The user should check that there are no constraint redundancies. If the data for the constraints are accurate only to the absolute precision σ, the user should ensure that the value of the optional parameter **ftol** is *greater* than σ. For example, if all elements of A are of order unity and are accurate only to three decimal places, the optional parameter **ftol** should be at least 10^{-3}.

NW_TOO_MANY_ITER

The maximum number of iterations, ⟨*value*⟩, have been performed.

The value of the optional parameter **max_iter** may be too small. If the method appears to be making progress (e.g., the objective function is being satisfactorily reduced), increase the value of **options.max_iter** and rerun nag_opt_qp (possibly using the **options.start** = **Nag_Warm** facility to specify the initial working set).

NE_HESS_TOO_BIG

Reduced Hessian exceeds assigned dimension. **options.max_df** = ⟨*value*⟩.

The algorithm needed to expand the reduced Hessian when it was already at its maximum dimension, as specified by the optional parameter **max_df**.

The value of the parameter **max_df** is too small. Rerun nag_opt_qp with a larger value (possibly using the **start** = **Nag_Warm** facility to specify the initial working set).

NW_OVERFLOW_WARN

Serious ill conditioning in the working set after adding constraint ⟨*value*⟩. Overflow may occur in subsequent iterations.

If overflow occurs preceded by this warning then serious ill conditioning has probably occurred in the working set when adding a constraint. It may be possible to avoid the difficulty by increasing the magnitude of the optional parameter **ftol** and re-running the program. If the message recurs even after this change, the offending linearly dependent constraint j must be removed from the problem.

NE_NOT_APPEND_FILE

Cannot open file ⟨*string*⟩ for appending.

NE_WRITE_ERROR

Error occurred when writing to file ⟨*string*⟩.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*string*⟩.

10. Further Comments

Sensible scaling of the problem is likely to reduce the number of iterations required and make the problem less sensitive to perturbations in the data, thus improving the condition of the problem. In the absence of better information it is usually sensible to make the Euclidean lengths of each constraint of comparable magnitude. See the Chapter Introduction and Gill *et al*(1986) for further information and advice.

10.1. Accuracy

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

11. References

Bunch J R and Kaufman L C (1980) A Computational Method for the Indefinite Quadratic Programming Problem *Linear Algebra and its Applications* **34** 341–370.

Gill P E, Hammarling S J, Murray W, Saunders M A and Wright M H (1986) *User's Guide for LSSOL (Version 1.0): A Fortran Package for Constrained Least-squares and Convex Quadratic Programming* Report SOL 86-1, Department of Operations Research, Stanford University.

Gill P E and Murray W (1978) Numerically Stable Methods for Quadratic Programming *Mathematical Programming* **14** 349–372.

Gill P E, Murray W, Saunders M A and Wright M H (1984) Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints *ACM Trans. Math. Softw.* **10** 282–298.

Gill P E, Murray W, Saunders M A and Wright M H (1989) A Practical Anti-cycling Procedure for Linearly Constrained Optimization *Mathematical 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 Review* **33** 1–36.

Pardalos P M and Schnitger G (1988) Checking Local Optimality in Constrained Quadratic Programming is NP-hard *Operations Research Letters* **7** 33–35.

12. See Also

nag_opt_lp (e04mfc)
nag_opt_init (e04xxc)
nag_opt_read (e04xyc)
nag_opt_free (e04xzc)

13. Example 2

To minimize the quadratic function $f(x) = c^T x + \frac{1}{2} x^T H x$, where

$$c = (-0.02,\ -0.2,\ -0.2,\ -0.2,\ -0.2,\ 0.04,\ 0.04)^T$$

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

subject to the bounds

$$\begin{aligned} -0.01 &\le x_1 \le 0.01 \\ -0.10 &\le x_2 \le 0.15 \\ -0.01 &\le x_3 \le 0.03 \\ -0.04 &\le x_4 \le 0.02 \\ -0.10 &\le x_5 \le 0.05 \\ -0.01 &\le x_6 \\ -0.01 &\le x_7 \end{aligned}$$

and the general constraints

$$\begin{array}{rl} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = -0.13 \\ & 0.15x_1 + 0.04x_2 + 0.02x_3 + 0.04x_4 + 0.02x_5 + 0.01x_6 + 0.03x_7 \le -0.0049 \\ & 0.03x_1 + 0.05x_2 + 0.08x_3 + 0.02x_4 + 0.06x_5 + 0.01x_6 \le -0.0064 \\ & 0.02x_1 + 0.04x_2 + 0.01x_3 + 0.02x_4 + 0.02x_5 \le -0.0037 \\ & 0.02x_1 + 0.03x_2 + 0.01x_5 \le -0.0012 \\ -0.0992 \le & 0.70x_1 + 0.75x_2 + 0.80x_3 + 0.75x_4 + 0.80x_5 + 0.97x_6 \\ -0.003 \le & 0.02x_1 + 0.06x_2 + 0.08x_3 + 0.12x_4 + 0.02x_5 + 0.01x_6 + 0.97x_7 \le 0.002 \end{array}$$

The initial point, which is infeasible, is

$$x_0 = (-0.01,\ -0.03,\ 0.0,\ -0.01,\ -0.1,\ 0.02,\ 0.01)^T.$$

The computed solution (to five figures) is

$$x^* = (-0.01,\ -0.069865,\ 0.018259,\ -0.024261,\ -0.062006,\ 0.013054,\ 0.0040665)^T.$$

One bound constraint and four general constraints are active at the solution.

This example shows the use of certain optional parameters. Option values are assigned directly within the program text and by reading values from a data file. The **options** structure is declared and initialized by nag_opt_init (e04xxc), a value is then assigned directly to option **inf_bound** and two further options are read from the data file by use of nag_opt_read (e04xyc). nag_opt_qp is then called to solve the problem using the function `qphess2`, with the Hessian implicit, for argument

qphess. On successful return two further options are set, selecting a warm start and a reduced level of printout, and the problem is solved again using the function `qphess3`. In this case the Hessian is defined explicitly. Finally the memory freeing function nag_opt_free (e04xzc) is used to free the memory assigned to the pointers in the options structure. Users should **not** use the standard C function `free()` for this purpose.

13.1. Program Text

```
static void ex2()
{
  double x[MAXN], cvec[MAXN];
  double a[MAXLIN][MAXN], h[MAXN][MAXN];
  double bl[MAXBND], bu[MAXBND];
  double objf;
  Integer tda, tdh;
  Integer i, j, n, nclin, nbnd;
  Boolean print;
  Nag_E04_Opt options;
  static NagError fail, fail2;

  Vprintf("\nExample 2: some optional parameters are set.\n");
  Vscanf(" %*[^\n]"); /* Skip heading in data file */

  fail.print = TRUE;
  fail2.print = TRUE;

  /* Set the actual problem dimensions.
   * n = the number of variables.
   * nclin = the number of general linear constraints (may be 0).
   */
  tda = MAXN;
  tdh = MAXN;
  n = 7;
  nclin =  7;

  /* cvec   = the coefficients of the explicit linear term of f(x).
   * a      = the linear constraint matrix.
   * bl     = the lower bounds on x and A*x.
   * bu     = the upper bounds on x and A*x.
   * x      = the initial estimate of the solution.
   */

  /* Read the coefficients of the explicit linear term of f(x). */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < n; ++i)
    Vscanf("%lf",&cvec[i]);

  /* Read the linear constraint matrix A. */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nclin; ++i)
    for (j = 0; j < n; ++j)
      Vscanf("%lf",&a[i][j]);

  /* Read the bounds. */
  nbnd = n + nclin;
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nbnd; ++i)
    Vscanf("%lf", &bl[i]);
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nbnd; ++i)
    Vscanf("%lf", &bu[i]);

  /* Read the initial estimate of x. */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < n; ++i)
    Vscanf("%lf",&x[i]);

  e04xxc(&options); /* Initialise options structure */

  /* Set one option directly
```

```
   * Bounds  >=   inf_bound will be treated as plus  infinity.
   * Bounds  <=  -inf_bound will be treated as minus infinity.
   */
  options.inf_bound = 1.0e21;

  /* Read remaining option values from file */
  fail.print = TRUE;
  print = TRUE;
  e04xyc("e04nfc", "stdin", &options, print, "stdout", &fail);

  /* Solve the problem from a cold start.
   * The Hessian is defined implicitly by function qphess2.
   */
  if (fail.code == NE_NOERROR)
    e04nfc(n, nclin, (double *)a, tda, bl, bu, cvec, (double *)0, tdh,
           qphess2, x, &objf, &options, NAGCOMM_NULL, &fail);

  if (fail.code == NE_NOERROR)
    {
      /* The following is for illustrative purposes only. We do a warm
       * start with the final working set of the previous run.
       * This time we store the Hessian explicitly in h[][], and use
       * the corresponding function qphess3().
       * Only the final solution from the results is printed.
       */
      Vprintf("\nA run of the same example with a warm start:\n");

      options.start = Nag_Warm;
      options.print_level = Nag_Soln;

      for (i = 0; i < n; ++i)
        {
          for (j = 0; j < n; ++j) h[i][j] = 0.0;
          if (i <= 4) h[i][i] = 2.0;
          else h[i][i] = -2.0;
        }
      h[2][3] = 2.0;
      h[3][2] = 2.0;
      h[5][6] = -2.0;
      h[6][5] = -2.0;

      /* Solve the problem again. */
      e04nfc(n, nclin, (double *)a, tda, bl, bu, cvec, (double *)h, tdh,
             qphess3, x, &objf, &options, NAGCOMM_NULL, &fail);
    }
  /* Free memory allocated by e04nfc to pointers in options */
  e04xzc(&options, "all", &fail2);

  if (fail.code != NE_NOERROR || fail2.code != NE_NOERROR) exit(EXIT_FAILURE);
} /* ex2 */


static void qphess2(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm)
{
  /* In this version of qphess the Hessian matrix is implicit.
   * The array h[] is not accessed. There is no special coding
   * for the case jthcol > 0.
   */

  hx[0] = 2.0*x[0];
  hx[1] = 2.0*x[1];
  hx[2] = 2.0*(x[2] + x[3]);
  hx[3] = hx[2];
  hx[4] = 2.0*x[4];
  hx[5] = -2.0*(x[5] + x[6]);
  hx[6] = hx[5];
} /* qphess2 */
```

```
static void qphess3(Integer n, Integer jthcol, double h[], Integer tdh,
                    double x[], double hx[], Nag_Comm *comm)
{
  /* In this version of QPHESS, the matrix H is stored in h[]
   * as a full two-dimensional array.
   */

#define H(I,J) h[(I)*tdh + (J)]

  Integer i, j;

  if (jthcol != 0)
    {
      /* Special case -- extract one column of  H. */
      j = jthcol - 1;
      for (i = 0; i < n; ++i)
        hx[i] = H(i,j);
    }
  else
    {
      /* Normal Case. */
      for (i = 0; i < n; ++i) hx[i] = 0.0;

      for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
          hx[i] += H(i,j)*x[j];
    }
} /* qphess3 */
```

13.2. Program Data

```
e04nfc Example Program Data

Linear term of f(x), c.
-0.02 -0.2 -0.2 -0.2 -0.2 0.04 0.04

Linear constraint matrix, A.
1.0  1.0  1.0  1.0  1.0  1.0  1.0
0.15 0.04 0.02 0.04 0.02 0.01 0.03
0.03 0.05 0.08 0.02 0.06 0.01 0.0
0.02 0.04 0.01 0.02 0.02 0.0  0.0
0.02 0.03 0.0  0.0  0.01 0.0  0.0
0.70 0.75 0.80 0.75 0.80 0.97 0.0
0.02 0.06 0.08 0.12 0.02 0.01 0.97

Lower bounds
-0.01 -0.1 -0.01 -0.04 -0.1 -0.01 -0.01
-0.13 -1.0e21 -1.0e21 -1.0e21 -1.0e21 -0.0992 -0.003

Upper bounds
 0.01  0.15 0.03  0.02  0.05 1.0e21 1.0e21
-0.13 -0.0049 -0.0064 -0.0037 -0.0012  1.0e21  0.002

Initial estimate of x
-0.01 -0.03 0.0 -0.01 -0.1 0.02 0.01

Following options for e04nfc are read by e04xyc in example 2.

begin e04nfc

 fmax_iter = 30 /* Set maximum number of iterations in feasiblity phase */
 max_iter = 50  /* Set maximum total number of iterations */

end
```

13.3. Program Results

```
Example 2: some optional parameters are set.

Optional parameter setting for e04nfc.
--------------------------------------

Option file: stdin

fmax_iter set to 30
max_iter set to 50

Parameters to e04nfc
--------------------

Linear constraints............   7    Number of variables...........   7

prob....................  Nag_QP2    start...................  Nag_Cold
ftol....................  1.05e-08   reset_ftol..............         5
rank_tol................  1.11e-14   crash_tol...............  1.00e-02
fcheck..................        50   max_df..................         7
inf_bound...............  1.00e+21   inf_step................  1.00e+21
fmax_iter...............        30   max_iter................        50
hrows...................         7   machine precision.......  1.11e-16
optim_tol...............  1.72e-13   min_infeas..............     FALSE
print_level......... Nag_Soln_Iter
outfile.................    stdout

Memory allocation:
state...................       Nag
ax......................       Nag    lambda..................       Nag

Results from e04nfc:
-------------------

  Itn Jdel  Jadd   Step    Ninf  Sinf/Obj    Bnd  Lin  Nart  Nrz  Norm Gz

   0   0     0    0.0e+00   3   1.0380e-01    3    4    0     0   0.00e+00
   1   9 U  13 L  4.1e-02   1   3.0000e-02    3    4    0     0   0.00e+00
   2  12 U   4 L  4.2e-02   0   0.0000e+00    4    3    0     0   0.00e+00

 Itn 2  -- Feasible point found.
   2   0     0    0.0e+00   0   4.5800e-02    4    3    0     0   0.00e+00
   3   5 L  14 L  1.3e-01   0   4.1616e-02    3    4    0     0   0.00e+00
   4  11 U   0    1.0e+00   0   3.9362e-02    3    3    0     1   4.16e-17
   5   3 L  10 U  4.1e-01   0   3.7589e-02    2    4    0     1   1.19e-02
   6   0     0    1.0e+00   0   3.7554e-02    2    4    0     1   1.04e-17
   7   4 L   0    1.0e+00   0   3.7032e-02    1    4    0     2   3.80e-17

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V    1    LL  -1.00000e-02  -1.0000e-02   1.0000e-02    4.700e-01   0.000e+00
 V    2    FR  -6.98646e-02  -1.0000e-01   1.5000e-01    0.000e+00   3.014e-02
 V    3    FR   1.82592e-02  -1.0000e-02   3.0000e-02    0.000e+00   1.174e-02
 V    4    FR  -2.42608e-02  -4.0000e-02   2.0000e-02    0.000e+00   1.574e-02
 V    5    FR  -6.20056e-02  -1.0000e-01   5.0000e-02    0.000e+00   3.799e-02
 V    6    FR   1.38054e-02  -1.0000e-02   None       0.000e+00   2.381e-02
 V    7    FR   4.06650e-03  -1.0000e-02   None       0.000e+00   1.407e-02

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L    1    EQ  -1.30000e-01  -1.3000e-01  -1.3000e-01   -1.908e+00   2.776e-17
 L    2    FR  -5.87990e-03     None      -4.9000e-03    0.000e+00   9.799e-04
 L    3    UL  -6.40000e-03     None      -6.4000e-03   -3.144e-01   0.000e+00
 L    4    FR  -4.53732e-03     None      -3.7000e-03    0.000e+00   8.373e-04
 L    5    FR  -2.91600e-03     None      -1.2000e-03    0.000e+00   1.716e-03
 L    6    LL  -9.92000e-02  -9.9200e-02   None       1.955e+00   1.388e-17
 L    7    LL  -3.00000e-03  -3.0000e-03   2.0000e-03    1.972e+00  -4.337e-19
```

```
Exit after 7 iterations.

Optimal QP solution found.

Final QP objective value =   3.7031646e-02


A run of the same example with a warm start:

Parameters to e04nfc
--------------------

Linear constraints............   7    Number of variables...........   7

prob....................  Nag_QP2    start...................  Nag_Warm
ftol....................  1.05e-08    reset_ftol..............         5
rank_tol................  1.11e-14    crash_tol...............  1.00e-02
fcheck..................        50    max_df..................         7
inf_bound...............  1.00e+21    inf_step................  1.00e+21
fmax_iter...............        30    max_iter................        50
hrows...................         7    machine precision.......  1.11e-16
optim_tol...............  1.72e-13    min_infeas..............     FALSE
print_level.........      Nag_Soln
outfile.................    stdout

Memory allocation:
state...................       Nag
ax......................       Nag    lambda..................       Nag

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1    LL  -1.00000e-02  -1.0000e-02   1.0000e-02    4.700e-01   0.000e+00
 V   2    FR  -6.98646e-02  -1.0000e-01   1.5000e-01    0.000e+00   3.014e-02
 V   3    FR   1.82592e-02  -1.0000e-02   3.0000e-02    0.000e+00   1.174e-02
 V   4    FR  -2.42608e-02  -4.0000e-02   2.0000e-02    0.000e+00   1.574e-02
 V   5    FR  -6.20056e-02  -1.0000e-01   5.0000e-02    0.000e+00   3.799e-02
 V   6    FR   1.38054e-02  -1.0000e-02   None       0.000e+00   2.381e-02
 V   7    FR   4.06650e-03  -1.0000e-02   None       0.000e+00   1.407e-02

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1    EQ  -1.30000e-01  -1.3000e-01  -1.3000e-01   -1.908e+00   0.000e+00
 L   2    FR  -5.87990e-03     None      -4.9000e-03    0.000e+00   9.799e-04
 L   3    UL  -6.40000e-03     None      -6.4000e-03   -3.144e-01   0.000e+00
 L   4    FR  -4.53732e-03     None      -3.7000e-03    0.000e+00   8.373e-04
 L   5    FR  -2.91600e-03     None      -1.2000e-03    0.000e+00   1.716e-03
 L   6    LL  -9.92000e-02  -9.9200e-02   None       1.955e+00   0.000e+00
 L   7    LL  -3.00000e-03  -3.0000e-03   2.0000e-03    1.972e+00  -1.735e-18

Exit after 0 iterations.

Optimal QP solution found.

Final QP objective value =   3.7031646e-02
```