NAG Library Function Document

nag_opt_sparse_convex_qp_option_set_file (e04nrc)

1 Purpose

nag_opt_sparse_convex_qp_option_set_file (e04nrc) may be used to supply optional arguments to nag_opt_sparse_convex_qp_solve (e04nqc) from an external file. The initialization function nag_opt_sparse_convex_qp_init (e04npc) **must** have been called before calling nag_opt_sparse_convex_qp_option_set_file (e04nrc).

2 Specification

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

```
void nag_opt_sparse_convex_qp_option_set_file (Nag_FileID fileid,
     Nag_E04State *state, NagError *fail)
```

3 Description

nag_opt_sparse_convex_qp_option_set_file (e04nrc) may be used to supply values for optional arguments to nag_opt_sparse_convex_qp_solve (e04nqc). nag_opt_sparse_convex_qp_option_set_file (e04nrc) reads an external file whose **fileid** has been returned by a call to nag_open_file (x04acc). nag_open_file (x04acc) must be called to provide **fileid**. Each line of the file defines a single optional argument. It is only necessary to supply values for those arguments whose values are to be different from their default values.

Each optional argument is defined by a single character string, consisting of one or more items. The items associated with a given option must be separated by spaces, or equals signs [=]. Alphabetic characters may be upper or lower case. The string

```
Print Level = 1
```

is an example of a string used to set an optional argument. For each option the string contains one or more of the following items:

- a mandatory keyword;
- a phrase that qualifies the keyword;
- a number that specifies an Integer or double value. Such numbers may be up to 16 contiguous characters which can be read using C's `d` or `g` formats, terminated by a space if this is not the last item on the line.

Blank strings and comments are ignored. A comment begins with an asterisk (*) and all subsequent characters in the string are regarded as part of the comment.

The file containing the options must start with `Begin` and must finish with `End`. An example of a valid options file is:

```
Begin * Example options file
   Print level = 5
End
```

Optional argument settings are preserved following a call to nag_opt_sparse_convex_qp_solve (e04nqc) and so the keyword **Defaults** is provided to allow you to reset all the optional arguments to their default values before a subsequent call to nag_opt_sparse_convex_qp_solve (e04nqc).

A complete list of optional arguments, their abbreviations, synonyms and default values is given in Section 12 in nag_opt_sparse_convex_qp_solve (e04nqc).

4 References

None.

5 Arguments

1: **fileid** – Nag_FileID *Input*

On entry: the ID of the option file to be read as returned by a call to nag_open_file (x04acc).

2: **state** – Nag_E04State * *Communication Structure*

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

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument $\langle \mathit{value} \rangle$ had an illegal value.

NE_E04NPC_NOT_INIT

Initialization function nag_opt_sparse_convex_qp_init (e04npc) has not been called.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_OPTIONS_FILE_READ_FAILURE

At least one line of the options file is invalid.

Could not read options file on unit **fileid** $= \langle \mathit{value} \rangle$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_opt_sparse_convex_qp_option_set_string (e04nsc), nag_opt_sparse_convex_qp_option_set_integer (e04ntc) or nag_opt_sparse_convex_qp_option_set_double (e04nuc) may also be used to supply optional arguments to nag_opt_sparse_convex_qp_solve (e04nqc).

10 Example

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

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

and

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

subject to the bounds

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

and to the linear constraints

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

The initial point, which is infeasible, is

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

The optimal solution (to five figures) is

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

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

10.1 Program Text

```
/* nag_opt_sparse_convex_qp_option_set_file (e04nrc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 8, 2004.
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL qphx(Integer ncolh, const double x[], double hx[],
                          Integer nstate, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  const char    *optionsfile = "e04nrce.opt";
  Integer       exit_status = 0;
```

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

/* Arrays */
static double ruser[1] = {-1.0};
char         nag_enum_arg[40];
char         *cuser = 0, *prob = 0;
char         **names = 0;
double       *acol = 0, *bl = 0, *bu = 0, *c = 0, *pi = 0, *rc = 0;
double       *x = 0;
Integer      *helast = 0, *hs = 0, *inda = 0, *iuser = 0, *loca = 0;

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

INIT_FAIL(fail);

printf("%s", "nag_opt_sparse_convex_qp_option_set_file (e04nrc) Example"
       " Program Results");
printf("\n");

/* For communication with user-supplied functions: */
comm.user = ruser;

/* This program demonstrates the use of routines to set and
 * get values of optional parameters associated with
 * nag_opt_sparse_convex_qp_solve (e04nqc).
 */

/* Skip heading in data file. */
scanf("%*[^\n] ");
scanf("%ld %ld ", &n, &m);
scanf("%*[^\n] ");

if (n >= 1 && m >= 1)
  {
    /* Read ne, iobj, ncolh, start and nname from data file. */
    scanf("%ld %ld %ld %39s %ld",
          &ne, &iobj, &ncolh, nag_enum_arg, &nname);
    scanf("%*[^\n] ");
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    start = (Nag_Start) nag_enum_name_to_value(nag_enum_arg);

    /* Allocate memory */
    if (!(names = NAG_ALLOC(n+m, char *)) ||
        !(prob = NAG_ALLOC(9, char)) ||
        !(acol = NAG_ALLOC(ne, double)) ||
        !(bl = NAG_ALLOC(m+n, double)) ||
        !(bu = NAG_ALLOC(m+n, double)) ||
        !(c = NAG_ALLOC(1, double)) ||
        !(pi = NAG_ALLOC(m, double)) ||
        !(rc = NAG_ALLOC(n+m, double)) ||
        !(x = NAG_ALLOC(n+m, double)) ||
        !(helast = NAG_ALLOC(n+m, Integer)) ||
        !(hs = NAG_ALLOC(n+m, Integer)) ||
        !(inda = NAG_ALLOC(ne, Integer)) ||
        !(iuser = NAG_ALLOC(1, Integer)) ||
        !(loca = NAG_ALLOC(n+1, Integer)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
  }
else
  {
    printf("Invalid n or nf or nea or neg\n");
    exit_status = 1;
    goto END;
  }
/* Read names from data file. */

for (i = 1; i <= nname; ++i)
  {
    names[i-1] = NAG_ALLOC(9, char);
    scanf(" ' %8s '", names[i-1]);
  }
scanf("%*[^\n] ");

/* Read the matrix acol from data file. Set up loca. */
jcol = 1;
loca[jcol - 1] = 1;
for (i = 1; i <= ne; ++i)
  {
    /* Element ( inda[i-1], icol ) is stored in acol[i-1]. */
    scanf("%lf %ld %ld", &acol[i - 1], &inda[i - 1], &icol);
    scanf("%*[^\n] ");

    if (icol < jcol)
      {
        /* Elements not ordered by increasing column index. */
        printf("%s %5ld %s %5ld",
               "Element in column", icol,
               " found after element in column", jcol);
        printf("%s\n\n", ". Problem abandoned.");
      }
    else if (icol == jcol + 1)
      {
        /* Index in acol of the start of the icol-th column equals i. */
        loca[icol - 1] = i;
        jcol = icol;
      }
    else if (icol > jcol + 1)
      {
        /* Index in acol of the start of the icol-th column equals i,
         * but columns jcol+1,jcol+2,...,icol-1 are empty. Set the
         * corresponding elements of loca to i.
         */
        for (j = jcol + 1; j <= icol - 1; ++j)
          {
            loca[j - 1] = i;
          }
        loca[icol - 1] = i;
        jcol = icol;
      }
  }
loca[n] = ne + 1;
if (n > icol)
  {
    /* Columns n,n-1,...,icol+1 are empty. Set the corresponding */
    /* elements of loca accordingly. */
    for (i = n; i >= icol + 1; --i)
      {
        loca[i - 1] = loca[i];
      }
  }

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

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

if (start == Nag_Cold)
  {
    for (i = 1; i <= n; ++i)
      {
        scanf("%ld", &hs[i - 1]);
      }
    scanf("%*[^\n] ");
  }
else if (start == Nag_Warm)
  {
    for (i = 1; i <= n + m; ++i)
      {
        scanf("%ld", &hs[i - 1]);
      }
    scanf("%*[^\n] ");
  }

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

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

/* nag_opt_sparse_convex_qp_init (e04npc).
 * Initialization function for
 * nag_opt_sparse_convex_qp_solve (e04nqc)
 */
nag_opt_sparse_convex_qp_init(&state, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Initialisation of nag_opt_sparse_convex_qp_solve (e04nqc)"
           " failed.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* By default nag_opt_sparse_convex_qp_solve (e04nqc) does not print
 *  monitoring information. Call nag_open_file (x04acc) to set the print file
 *  fileidout
 */
/* nag_open_file (x04acc).
 * Open unit number for reading, writing or appending, and
 * associate unit with named file
 */
nag_open_file("", 2, &fileidout, &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 2;
    goto END;
  }
/* nag_opt_sparse_convex_qp_option_set_integer (e04ntc).
 * Set a single option for nag_opt_sparse_convex_qp_solve
 * (e04nqc) from an integer argument
 */
fflush(stdout);
nag_opt_sparse_convex_qp_option_set_integer("Print file", fileidout, &state,
                                            &fail);
```

```
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }
/* Set input to options file to read. */
/* nag_open_file (x04acc), see above. */
nag_open_file(optionsfile, 0, &fileidin, &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }
/* nag_opt_sparse_convex_qp_option_set_file (e04nrc).
 * Supply optional parameter values for
 * nag_opt_sparse_convex_qp_solve (e04nqc) from external
 * file
 */
nag_opt_sparse_convex_qp_option_set_file(fileidin, &state, &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }
nag_close_file(fileidin, &fail);               /* Close Library output */

/* Use nag_opt_sparse_convex_qp_option_get_integer (e04nxc) to find the value
 * of Integer-valued option 'Elastic mode'.
 */
/* nag_opt_sparse_convex_qp_option_get_integer (e04nxc).
 * Get the setting of an integer valued option of
 * nag_opt_sparse_convex_qp_solve (e04nqc)
 */
nag_opt_sparse_convex_qp_option_get_integer("Elastic mode", &elmode, &state,
                                            &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }
printf("Option 'Elastic mode' has the value %3ld.\n", elmode);
/* Use nag_opt_sparse_convex_qp_option_set_double (e04nuc) to set the value of
 *  real-valued option 'Infinite bound size'.
 */
bndinf = 1e10;
/* nag_opt_sparse_convex_qp_option_set_double (e04nuc).
 * Set a single option for nag_opt_sparse_convex_qp_solve
 * (e04nqc) from a double argument
 */
nag_opt_sparse_convex_qp_option_set_double("Infinite bound size", bndinf,
                                           &state, &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }

/* Use nag_opt_sparse_convex_qp_option_get_double (e04nyc) to find the value
 *  of real-valued option 'Feasibility tolerance'.
 */
/* nag_opt_sparse_convex_qp_option_get_double (e04nyc).
 * Get the setting of a double valued option of
 * nag_opt_sparse_convex_qp_solve (e04nqc)
 */
nag_opt_sparse_convex_qp_option_get_double("Feasibility tolerance", &featol,
                                           &state, &fail);
if (fail.code != NE_NOERROR)
  {
    exit_status = 1;
    goto END;
  }
  printf("Option 'Feasibility tolerance' has the value %14.5e.\n",
         featol);

  /* Use nag_opt_sparse_convex_qp_option_set_string (e04nsc) to set the option
   *  'Iterations limit'.
   */
  /* nag_opt_sparse_convex_qp_option_set_string (e04nsc).
   * Set a single option for nag_opt_sparse_convex_qp_solve
   * (e04nqc) from a character string
   */
  nag_opt_sparse_convex_qp_option_set_string("Iterations limit 50", &state,
                                             &fail);
  if (fail.code != NE_NOERROR)
    {
      exit_status = 1;
      goto END;
    }
  fflush(stdout);

  /* Solve the QP problem. */
  /* nag_opt_sparse_convex_qp_solve (e04nqc).
   * LP or QP problem (suitable for sparse problems)
   */
  nag_open_file("", 2, &fileidout, &fail);
  fflush(stdout);
  nag_opt_sparse_convex_qp_solve(start, qphx, m, n, ne, nname, lenc, ncolh,
                                 iobj, objadd, prob, acol, inda, loca, bl, bu,
                                 c, (const char **) names, helast, hs, x, pi,
                                 rc,
                                 &ns, &ninf, &sinf, &obj, &state, &comm,
                                 &fail);

  if (fail.code == NE_NOERROR)
    {
      printf("Final objective value = %12.3e\n", obj);
      printf("Optimal X = ");

      for (i = 1; i <= n; ++i)
        {
          printf("%9.2f%s", x[i - 1], i%7 == 0 || i == n ? "\n" : " ");
        }
    }
  else
    {
      printf(
        "Error from nag_opt_sparse_convex_qp_solve (e04nqc).\n%s\n",
        fail.message);
      exit_status = 1;
      goto END;
    }

  if (fail.code != NE_NOERROR)
    {
      exit_status = 2;
    }

 END:
  NAG_FREE(cuser);
  for (i = 1; i <= nname; ++i)
    {
      NAG_FREE(names[i-1]);
    }
  NAG_FREE(names);
  NAG_FREE(prob);
  NAG_FREE(acol);
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(c);
  NAG_FREE(pi);
  NAG_FREE(rc);
  NAG_FREE(x);
  NAG_FREE(helast);
  NAG_FREE(hs);
  NAG_FREE(inda);
  NAG_FREE(iuser);
  NAG_FREE(loca);

  return exit_status;
}

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

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

  /* Function Body */
  if (comm->user[0] == -1.0)
    {
      fflush(stdout);
      printf("(User-supplied callback qphx, first invocation.)\n");
      comm->user[0] = 0.0;
      fflush(stdout);
    }
  HX(1) = X(1) * 2;
  HX(2) = X(2) * 2;
  HX(3) = (X(3) + X(4)) * 2;
  HX(4) = HX(3);
  HX(5) = X(5) * 2;
  HX(6) = (X(6) + X(7)) * 2;
  HX(7) = HX(6);
  return;
} /* qphx */
```

10.2 Program Data

```
nag_opt_sparse_convex_qp_option_set_file (e04nrc) Example Program Data
 7  8                       : Values of n and m
48  8  7  Nag_Cold  15      : Values of nnz, iobj, ncolh, start and nname

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

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

```
-2000.00   8   3
    1.00   1   4
    0.12   7   4
    0.02   3   4
    0.02   4   4
    0.75   6   4
    0.04   2   4
-2000.00   8   4
    0.01   5   5
    0.80   6   5
    0.02   7   5
    1.00   1   5
    0.02   2   5
    0.06   3   5
    0.02   4   5
-2000.00   8   5
    1.00   1   6
    0.01   2   6
    0.01   3   6
    0.97   6   6
    0.01   7   6
  400.00   8   6
    0.97   7   7
    0.03   2   7
    1.00   1   7
  400.00   8   7        : End of matrix A

 0.0       0.0       4.0E+02   1.0E+02   0.0       0.0
 0.0       2.0E+03  -1.0E+25  -1.0E+25  -1.0E+25  -1.0E+25
 1.5E+03   2.5E+02  -1.0E+25             : End of lower bounds array BL

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

 0    0    0    0    0    0    0         : Initial array HS
 0.0  0.0  0.0  0.0  0.0  0.0  0.0       : Initial vector X
Begin nag_opt_sparse_convex_qp_option_set_file (e04nrc) example options
* Comment lines like this begin with an asterisk.
* Switch off output of timing information:
Timing level 0
* Allow elastic variables:
Elastic mode 1
* Set the feasibility tolerance:
Feasibility tolerance 1.0D-4
End
```

10.3 Program Results

```
nag_opt_sparse_convex_qp_option_set_file (e04nrc) Example Program Results


 OPTIONS file
 ------------

      Begin nag_opt_sparse_convex_qp_option_set_file (e04nrc) example options
      * Comment lines like this begin with an asterisk.
      * Switch off output of timing information:
      Timing level 0
      * Allow elastic variables:
      Elastic mode 1
      * Set the feasibility tolerance:
      Feasibility tolerance 1.0D-4
      End
Option 'Elastic mode' has the value   1.
Option 'Feasibility tolerance' has the value   1.00000e-04.

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

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

 LP/QP Parameters
 ----------------
 Minimize...............                 QPsolver Cholesky......                 Cold start.............
 Scale tolerance........     0.900       Feasibility tolerance..  1.00E-04       Iteration limit........        50
 Scale option...........         2       Optimality tolerance...  1.00E-06       Print level............         1
 Crash tolerance........     0.100       Pivot tolerance........  2.05E-11       Partial price..........         1
 Crash option...........         3       Elastic weight.........  1.00E+00       Prtl price section ( A)         7
 Elastic mode...........         1       Elastic objective......         1       Prtl price section (-I)         8

 QP objective
 ------------
 Objective variables....         7       Hessian columns........         7       Superbasics limit......         7
 Nonlin Objective vars..         7       Unbounded step size....  1.00E+10
 Linear Objective vars..         0

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



 Matrix statistics
 -----------------
               Total      Normal        Free       Fixed     Bounded
 Rows              8           5           1           1           1
 Columns           7           2           0           0           5

 No. of matrix elements                   48     Density      85.714
 Biggest                          1.0000E+00  (excluding fixed columns,
 Smallest                         1.0000E-02   free rows, and RHS)

 No. of objective coefficients             7
 Biggest                          2.0000E+03  (excluding fixed columns)
 Smallest                         2.0000E+02

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

(User-supplied callback qphx, first invocation.)

 Itn      1: Feasible linear constraints

 E04NQT EXIT   0 -- finished successfully
 E04NQT INFO   1 -- optimality conditions satisfied

 Problem name
 No. of iterations                   9   Objective value     -1.8477846771E+06
 No. of Hessian products            16   Objective row       -2.9886903537E+06
                                         Quadratic objective  1.1409056766E+06
 No. of superbasics                  2   No. of basic nonlinears             4
 No. of degenerate steps             0   Percentage                       0.00
 Max x       (scaled)        3 1.7E+00   Max pi      (scaled)        6 6.6E+06
 Max x                       3 6.5E+02   Max pi                      7 1.5E+04
 Max Prim inf(scaled)        0 0.0E+00   Max Dual inf(scaled)        4 2.4E-09
 Max Primal infeas           0 0.0E+00   Max Dual infeas             9 1.8E-11
 Name                                    Objective Value     -1.8477846771E+06

 Status         Optimal Soln             Iteration      9    Superbasics     2

 Section 1 - Rows

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

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

 Section 2 - Columns

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

       1  ...X1...    LL           .          -200.00000          .           200.00000      2360.67253      9
       2  ...X2...    BS       349.39923     -1301.20153          .          2500.00000         0.00000     10
       3  ...X3...   SBS       648.85342      -356.59829       400.00000       800.00000         0.00000     11
       4  ...X4...   SBS       172.84743      -356.59829       100.00000       700.00000         0.00000     12
       5  ...X5...    BS       407.52089     -1184.95822          .          1500.00000         0.00000     13
       6  ...X6...    BS       271.35624      1242.75804          .                None         0.00000     14
       7  ...X7...    BS       150.02278      1242.75804          .                None         0.00000     15
Final objective value =  -1.848e+06
Optimal X =      0.00    349.40    648.85    172.85    407.52    271.36    150.02
```