NAG Library Function Document

nag_opt_miqp_mps_write (e04mwc)

1 Purpose

nag_opt_miqp_mps_write (e04mwc) writes data for sparse linear programming, mixed integer linear programming, quadratic programming or mixed integer quadratic programming problems to a file in MPS format.

2 Specification

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

```
void nag_opt_miqp_mps_write (Nag_FileID fileid, Integer n, Integer m,
     Integer nnzc, Integer nnza, Integer ncolh, Integer nnzh,
     Integer lintvar, const Integer idxc[], const double c[], Integer iobj,
     const double a[], const Integer irowa[], const Integer iccola[],
     const double bl[], const double bu[], char pnames[][9],
     char crname[][9], const double h[], const Integer irowh[],
     const Integer iccolh[], Integer minmax, const Integer intvar[],
     NagError *fail)
```

3 Description

nag_opt_miqp_mps_write (e04mwc) writes data for linear programming (LP) or quadratic programming (QP) problems (or their mixed integer variants) from an optimization problem to a MPS output file, see Section 3.1 in nag_opt_miqp_mps_read (e04mxc) for the format description. The problem is expected in the form

$$\underset{x}{\text{minimize}}\, c^{\mathrm{T}}x + \tfrac{1}{2}x^{\mathrm{T}}Hx \quad \text{subject to} \quad l \leq \left\{ \begin{matrix} x \\ Ax \end{matrix} \right\} \leq u.$$

Where n is the number of variables, m is the number of general linear constraints, A is the linear constraint matrix with dimension m by n, the vectors l and u are the lower and upper bounds, respectively. H is the Hessian matrix with dimension n by n, however, only leading **ncolh** columns might contain nonzero elements and the rest is assumed to be zero.

Note that the linear term of the objective function c might be supplied either as **c** or via **iobj**. If **c** is supplied then **idxc** contains the indices of the nonzero elements of sparse vector c, whereas if **iobj** is supplied (**iobj** > 0), row **iobj** of matrix A is a free row storing the nonzero elements of c.

Note: that this function uses fixed MPS format, see IBM (1971).

4 References

IBM (1971) MPSX – Mathematical programming system *Program Number 5734 XM4* IBM Trade Corporation, New York

5 Arguments

1: **fileid** – Nag_FileID *Input*

On entry: the ID of the file to store the problem data as returned by a call to nag_open_file (x04acc).

Constraint: **fileid** ≥ 0.

2: **n** – Integer *Input*

On entry: n, the number of variables in the problem.

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

3: **m** – Integer *Input*

On entry: m, the number of constraints in the problem. This is the number of rows in the linear constraint matrix A, including the free row (if any; see **iobj**).

Constraint: $\mathbf{m} \geq 0$.

4: **nnzc** – Integer *Input*

On entry: the number of nonzero elements in the sparse vector c.

If $\mathbf{nnzc} = 0$, the vector c is considered empty and the arrays **idxc** and **c** will not be referenced and may be **NULL**. In this case the linear term of the objective function, if any, might be provided via **iobj**.

Constraints:

$\mathbf{nnzc} \geq 0$;
if $\mathbf{nnzc} > 0$, $\mathbf{iobj} = 0$.

5: **nnza** – Integer *Input*

On entry: the number of nonzero elements in matrix A.

If $\mathbf{nnza} = 0$, matrix A is considered empty, arrays **a** and **irowa** will not be referenced and may be **NULL**, and **iccola** should be the array of 1.

Constraint: $\mathbf{nnza} \geq 0$.

6: **ncolh** – Integer *Input*

On entry: the number of leading nonzero columns of the Hessian matrix H.

If $\mathbf{ncolh} = 0$, the quadratic term H of the objective function is considered zero (e.g., LP problems), and arrays **h**, **irowh** and **iccolh** will not be referenced and may be **NULL**.

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

7: **nnzh** – Integer *Input*

On entry: the number of nonzero elements of the Hessian matrix H.

Constraints:

if $\mathbf{ncolh} > 0$, $\mathbf{nnzh} > 0$;
otherwise $\mathbf{nnzh} = 0$.

8: **lintvar** – Integer *Input*

On entry: the number of integer variables in the problem.

If $\mathbf{lintvar} = 0$, all variables are considered continuous and array **intvar** will not be referenced and may be **NULL**.

Constraint: $\mathbf{lintvar} \geq 0$.

9: **idxc**[**nnzc**] – const Integer *Input*
10: **c**[**nnzc**] – const double *Input*

On entry: the nonzero elements of sparse vector c. $\mathbf{idxc}[i-1]$ must contain the index of $\mathbf{c}[i-1]$ in the vector, for $i = 1, 2, \ldots, \mathbf{nnzc}$.

The elements are stored in ascending order.

Constraints:

$1 \leq$ **idxc**$[i-1] \leq$ **n**, for $i = 1, 2, \ldots,$ **nnzc**;
idxc$[i-1] <$ **idxc**$[i]$, for $i = 1, 2, \ldots,$ **nnzc**.

11: **iobj** – Integer *Input*

On entry: if **iobj** > 0, row **iobj** of A is a free row containing the nonzero coefficients of the linear terms of the objective function. In this case **nnzc** is set to 0.

If **iobj** $= 0$, there is no free row in A, and the linear terms might be supplied in array **c**.

Constraint: if **iobj** > 0, **nnzc** $= 0$.

12: **a**[**nnza**] – const double *Input*
13: **irowa**[**nnza**] – const Integer *Input*
14: **iccola**[**n** + **1**] – const Integer *Input*

On entry: the nonzero elements of matrix A in compressed column storage (see Section 2.1.3 in the f11 Chapter Introduction). Arrays **irowa** and **a** store the row indices and the values of the nonzero elements, respectively. The elements are sorted by columns and within each column in nondecreasing order. Duplicate entries are not allowed. **iccola** contains the (one-based) indices to the beginning of each column in **a** and **irowa**.

If **nnza** $= 0$, **a** and **irowa** are not referenced and may be **NULL**.

Constraints:

$1 \leq$ **irowa**$[i-1] \leq$ **m**, for $i = 1, 2, \ldots,$ **nnza**;
iccola$[0] = 1$;
iccola$[i-1] \leq$ **iccola**$[i]$, for $i = 1, 2, \ldots,$ **n**;
iccola[**n**] $=$ **nnza** $+ 1$.

15: **bl**[**n** + **m**] – const double *Input*
16: **bu**[**n** + **m**] – const double *Input*

On entry: **bl** and **bu** contains the lower bounds l and the upper bounds u, respectively.

The first **n** elements refer to the bounds for the variables x and the rest to the bounds for the linear constraints (including the objective row **iobj** if present).

To specify a nonexistent lower bound (i.e., $l_j = -\text{inf}$), set **bl**$[j-1] \leq -10^{20}$; to specify a nonexistent upper bound, set **bu**$[j-1] \geq 10^{20}$.

Constraints:

bl$[j-1] \leq$ **bu**$[j-1]$, for $j = 1, 2, \ldots,$ **n** + **m**;
bl$[j-1] < 10^{20}$, for $j = 1, 2, \ldots,$ **n** + **m**;
bu$[j-1] > -10^{20}$, for $j = 1, 2, \ldots,$ **n** + **m**;
if **iobj** > 0, **bl**[**iobj** + **n** $- 1] \leq -10^{20}$ and **bu**[**iobj** + **n** $- 1] \geq 10^{20}$.

17: **pnames**[**5**][9] – char *Input*

On entry: a set of names associated with the MPSX form of the problem.

The names can be composed only from ‘printable’ characters (ASCII codes between 32 and 127).

If any of the names are blank, the default name is used.

pnames[0]
Contains the name of the problem.

pnames[1]
Contains the name of the objective row if the objective is provided in **c** instead of **iobj** and all names **crname** are given. The name must be nonempty and unique. In all other cases **pnames**[1] is not used.

pnames[2]
Contains the name of the RHS set.

pnames[3]
Contains the name of the RANGE.

pnames[4]
Contains the name of the BOUNDS.

18: **crname**[**n** + **m**][9] – char *Input*

On exit: the names of all the variables and constraints in the problem in that order.

The names can be composed only from 'printable' characters and must be unique.

If **crname** ≤ 0, **crname** is not referenced and may be **NULL**.

19: **h**[**nnzh**] – const double *Input*
20: **irowh**[**nnzh**] – const Integer *Input*
21: **iccolh**[**ncolh** + **1**] – const Integer *Input*

On entry: the nonzero elements of the Hessian matrix H in compressed column storage (see Section 2.1.3 in the f11 Chapter Introduction). The Hessian matrix, H, is symmetric and its elements are stored in a lower triangular matrix.

Arrays **irowh** and **h** store the row indices and the values of the nonzero elements, respectively. The elements are sorted by columns and within each column in nondecreasing order. Duplicate entries are not allowed. **iccolh** contains the (one-based) indices to the beginning of each column in **h** and **irowh**.

If **ncolh** $= 0$, **h** is not referenced and may be **NULL**.

Constraints:

$1 \leq$ **irowh**$[i-1] \leq$ **ncolh**, for $i = 1, 2, \ldots,$ **nnzh**;
iccolh$[0] = 1$;
iccolh$[i-1] \leq$ **iccolh**$[i]$, for $i = 1, 2, \ldots,$ **ncolh**;
iccolh[**ncolh**] $=$ **nnzh** $+ 1$.

22: **minmax** – Integer *Input*

On entry: **minmax** defines the direction of optimization problem.

minmax $= -1$
Minimization.

minmax $= 1$
Maximization.

Constraint: **minmax** $= -1$ or 1.

23: **intvar**[**lintvar**] – const Integer *Input*

On entry: **intvar** contains the indices k of variables x_k which are defined as integers. Duplicate indices are not allowed.

If **lintvar** $= 0$, **intvar** is not referenced and may be **NULL**.

Constraint: $1 \leq$ **intvar**$[j-1] \leq$ **n**, for $j = 1, 2, \ldots,$ **lintvar**.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ARRAY_INPUT

On entry, **iccola**$[0] = \langle value \rangle$.
Constraint: **iccola**$[0] = 1$.

On entry, **iccola**[**n**] $= \langle value \rangle$ and **nnza** $= \langle value \rangle$.
Constraint: **iccola**[**n**] $=$ **nnza** $+ 1$.

On entry, **iccolh**$[0] = \langle value \rangle$.
Constraint: **iccolh**$[0] = 1$.

On entry, **iccolh**[**ncolh**] $= \langle value \rangle$ and **nnzh** $= \langle value \rangle$.
Constraint: **iccolh**[**ncolh**] $=$ **nnzh** $+ 1$.

On entry, **intvar**$[\langle value \rangle] =$ **intvar**$[\langle value \rangle] = \langle value \rangle$.
Constraint: all entries in **intvar** must be unique.

NE_BAD_PARAM

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

On entry, **minmax** $= \langle value \rangle$.
Constraint: **minmax** $= -1$ or 1.

NE_BOUND

On entry, $j = \langle value \rangle$ and **bl**$[j-1] = \langle value \rangle$, **bl**$[j-1]$ is incorrect.
Constraint: **bl**$[j-1] < 1\mathrm{e}+20$.

On entry, $j = \langle value \rangle$, **bl**$[j-1] = \langle value \rangle$ and **bu**$[j-1] = \langle value \rangle$ are incorrect.
Constraint: **bl**$[j-1] \leq$ **bu**$[j-1]$.

On entry, $j = \langle value \rangle$, **bl**$[j-1] = \langle value \rangle$ and **bu**$[j-1] = \langle value \rangle$, the integer variable j requires at least one bound finite.
Constraint: at least one of the following conditions must be met for integer variable j: **bl**$[j-1] > -1\mathrm{e}+20$, **bu**$[j-1] < 1\mathrm{e}+20$.

On entry, $j = \langle value \rangle$ and **bu**$[j-1] = \langle value \rangle$, **bu**$[j-1]$ is incorrect.
Constraint: **bu**$[j-1] > -1\mathrm{e}+20$.

NE_FILEID

On entry, **fileid** $= \langle value \rangle$.
Constraint: **fileid** ≥ 0.

NE_INT

On entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 0.

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

NE_INT_2

On entry, **lintvar** $= \langle value \rangle$.
Constraint: **lintvar** ≥ 0.

On entry, **nnza** $= \langle value \rangle$.
Constraint: **nnza** ≥ 0.

On entry, **nnzc** $= \langle value \rangle$.
Constraint: **nnzc** ≥ 0.

NE_INT_3

On entry, **ncolh** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $0 \leq$ **ncolh** $\leq$ **n**.

On entry, **ncolh** $= \langle value \rangle$ and **nnzh** $= \langle value \rangle$.
Constraint: if **ncolh** $= 0$, **nnzh** $= 0$.

On entry, **ncolh** $= \langle value \rangle$ and **nnzh** $= \langle value \rangle$.
Constraint: if **ncolh** > 0, **nnzh** > 0.

NE_INT_4

On entry, **iobj** $= \langle value \rangle$ and **m** $= \langle value \rangle$.
Constraint: $0 \leq$ **iobj** $\leq$ **m**.

On entry, **iobj** $= \langle value \rangle$ and **nnzc** $= \langle value \rangle$.
Constraint: at most one of **iobj** or **nnzc** may be nonzero.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_CS

On entry, $i = \langle value \rangle$, **irowa**$[i-1] = \langle value \rangle$ and **m** $= \langle value \rangle$.
Constraint: $1 \leq$ **irowa**$[i-1] \leq$ **m**.

On entry, $j = \langle value \rangle$, $i = \langle value \rangle$, **ncolh** $= \langle value \rangle$ and **irowh**$[i-1] = \langle value \rangle$
Constraint: $j \leq$ **irowh**$[i-1] \leq$ **ncolh** (within the lower triangle).

On entry, more than one element of **a** has row index $\langle value \rangle$ and column index $\langle value \rangle$.
Constraint: each element of **a** must have a unique row and column index.

On entry, more than one element of **h** has row index $\langle value \rangle$ and column index $\langle value \rangle$.
Constraint: each element of **h** must have a unique row and column index.

NE_MPS_ILLEGAL_NAME

On entry, **crname**$[j-1]$ for $j = \langle value \rangle$ has been already used.
Constraint: the names in **crname** must be unique.

The name specified in **pnames**[1] is empty or has been already used among row names.
Constraint: the names in **pnames**[1] must be unique and nonempty if **crname** is provided and **nnzc** > 0.

NE_MPS_PRINTABLE

On entry, **crname**$[j-1]$ for $j = \langle value \rangle$ is incorrect.
Constraint: the names in **crname** must consist only of printable characters.

On entry, **pnames**$[j-1]$ for $j = \langle value \rangle$ is incorrect.
Constraint: the names in **pnames** must consist only of printable characters.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_INCREASING

On entry, $j = \langle value \rangle$, **iccola**$[j-1] = \langle value \rangle$ and **iccola**$[j] = \langle value \rangle$, the values of **iccola** must be nondecreasing.
Constraint: **iccola**$[j-1] \leq$ **iccola**$[j]$.

On entry, $j = \langle value \rangle$, **iccolh**$[j-1] = \langle value \rangle$ and **iccolh**$[j] = \langle value \rangle$, the values of **iccolh** must be nondecreasing.
Constraint: **iccolh**$[j-1] \leq$ **iccolh**$[j]$.

NE_NOT_STRICTLY_INCREASING

On entry, $j = \langle value \rangle$, **idxc**$[j-1] = \langle value \rangle$ and **idxc**$[j] = \langle value \rangle$.
Constraint: **idxc**$[j-1] <$ **idxc**$[j]$.

NE_OBJ_BOUND

On entry, **iobj** $= \langle value \rangle$, **bl**$[j-1] = \langle value \rangle$ and **bu**$[j-1] = \langle value \rangle$, if **iobj** > 0 the bounds must be infinite.
Constraints: **bl**$[j-1] \leq -1\mathrm{e}+20$, **bu**$[j-1] \geq 1\mathrm{e}+20$.

NE_STATE_VAL

On entry, $j = \langle value \rangle$, **idxc**$[j-1] = \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $1 \leq$ **idxc**$[j-1] \leq$ **n**.

On entry, $j = \langle value \rangle$, **intvar**$[j-1] = \langle value \rangle$ and **lintvar** $= \langle value \rangle$.
Constraint: $1 \leq$ **intvar**$[j-1] \leq$ **lintvar**.

NE_WRITE_ERROR

An error occurred when writing to file.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_miqp_mps_write (e04mwc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example shows how to store an optimization problem to a file in MPS format after it has been solved by nag_opt_sparse_convex_qp_solve (e04nqc). The problem is a minimization of the quadratic function $f(x) = c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$, where

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

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

subject to the bounds

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

and to the linear constraints

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

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

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

The generated file is called e04mwce.mps.

10.1 Program Text

```
/* nag_opt_miqp_mps_write (e04mwc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

/* Make a typedef for convenience when allocating crname. */
typedef char e04mw_name[9];
```

```
int main(void) {
  /* Scalars and arrays */

  /* Scalars */
  Integer exit_status = 0;
  double obj, objadd, sinf;
  Integer i, icol, iobj, j, jcol, lenc, m, n, ncolh, ne, ninf, nnzh, minmax,
    lintvar;
  Integer nname, ns;

  /* output file */
  char fname[] = "e04mwce.mps";

  /* Arrays */
  char prob[9];
  char **names;
  char pnames[5][9] = {"", "", "", "", ""};
  char (*crname)[9] = 0;
  double *acol = 0, *bl = 0, *bu = 0, *c = 0, *pi = 0, *rc = 0, *x = 0, *h = 0,
    *ruser = 0;
  Integer *helast = 0, *hs = 0, *inda = 0, *loca = 0, *iccolh = 0, *irowh = 0,
    *iuser = 0;

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

  INIT_FAIL(fail);

  printf("nag_opt_miqp_mps_write (e04mwc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file. */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read ne, iobj, ncolh, nnzh and nname from data file. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &m);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &m);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT
          "%*[^\n] ", &ne, &iobj, &ncolh, &nnzh, &nname);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT
        "%*[^\n] ", &ne, &iobj, &ncolh, &nnzh, &nname);
#endif

  if (n >= 1 && m >= 1) {
    /* Allocate memory */
    if (!(names = NAG_ALLOC(n + m, char *)) ||
        !(acol = NAG_ALLOC(ne, double)) ||
        !(bl = NAG_ALLOC(m + n, double)) ||
        !(bu = NAG_ALLOC(m + n, double)) ||
        !(c = NAG_ALLOC(1, double)) ||
        !(pi = NAG_ALLOC(m, double)) ||
        !(rc = NAG_ALLOC(n + m, double)) ||
        !(x = NAG_ALLOC(n + m, double)) ||
        !(helast = NAG_ALLOC(n + m, Integer)) ||
        !(hs = NAG_ALLOC(n + m, Integer)) ||
        !(inda = NAG_ALLOC(ne, Integer)) ||
        !(loca = NAG_ALLOC(n + 1, Integer)) ||
        !(iccolh = NAG_ALLOC(ncolh+1, Integer)) ||
        !(irowh = NAG_ALLOC(nnzh, Integer)) ||
        !(h = NAG_ALLOC(nnzh, double)) ||
        !(crname = NAG_ALLOC(n + m, e04mw_name)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("%s", "Either m or n invalid\n");
    exit_status = 1;
    return exit_status;
  }

  /* Read names fron file */
  for (i = 0; i < nname; ++i) {
    if (!(names[i] = NAG_ALLOC(9, char)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
#ifdef _WIN32
    scanf_s(" ' %8s '", names[i], 9);
#else
    scanf(" ' %8s '", names[i]);
#endif
  }

#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read the matrix acol from data file. Set up loca. */
  jcol = 0;
  loca[jcol] = 1;
  for (i = 0; i < ne; ++i) {
    /* Element (inda[i], icol) is stored in acol[i]. */
#ifdef _WIN32
    scanf_s("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &acol[i], &inda[i], &icol);
#else
    scanf("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &acol[i], &inda[i], &icol);
#endif
    if (icol <= jcol) {
      /* Elements not ordered by increasing column index. */
      printf("%s%5" NAG_IFMT "%s%5" NAG_IFMT "%s%s\n", "Element in column",
             icol, " found after element in column", jcol + 1, ". Problem",
             " abandoned.");
    }
    while (jcol + 1 < icol) {
      /* Fill in loca[] for missing columns */
      jcol++;
      loca[jcol] = i + 1;
    }
  }
  while (jcol < n) {
    /* Fill in loca[] for remaining columns */
    jcol++;
    loca[jcol] = i + 1;
  }

  /* Read the matrix h, from data file. Set up iccolh */
  jcol = 0;
  iccolh[jcol] = 1;
  for (i = 0; i < nnzh; ++i) {
    /* Element (irowh[i], icol) is stored in h[i]. */
#ifdef _WIN32
```

```
    scanf_s("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &h[i], &irowh[i],
          &icol);
#else
    scanf("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &h[i], &irowh[i],
          &icol);
#endif
    if (icol <= jcol) {
      /* Elements not ordered by increasing column index. */
      printf("%s%5" NAG_IFMT "%s%5" NAG_IFMT "%s%s\n", "Element in column",
             icol, " found after element in column", jcol, ". Problem",
             " abandoned.");
    }
    while (jcol + 1 < icol) {
      /* Fill in iccolh[] for missing columns */
      jcol++;
      iccolh[jcol] = i + 1;
    }
  }
  while (jcol < ncolh) {
    /* Fill in iccolh[] for remaining columns */
    jcol++;
    iccolh[jcol] = i + 1;
  }

  /* Read lower and upper bound */
  for (i = 0; i < n + m; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &bl[i]);
#else
    scanf("%lf", &bl[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < n + m; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &bu[i]);
#else
    scanf("%lf", &bu[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Set cold start and hs*/
  start = Nag_Cold;
  for (i = 0; i < n; ++i) {
    hs[i] = 0.0;
  }

  /* Read the initial point x_0 */
  for (i = 0; i < n; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
```

```
  /* 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("Initialization 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
     fileid

     nag_open_file (x04acc).
     Open unit number for reading, writing or appending, and
     associate unit with named file */
  nag_open_file("", 2, &fileid, NAGERR_DEFAULT);

  /* nag_opt_sparse_convex_qp_option_set_integer (e04ntc).
     Set a single option for nag_opt_sparse_convex_qp_solve (e04nqc)
     from an integer argument */
  nag_opt_sparse_convex_qp_option_set_integer("Print file", fileid, &state,
                                               &fail);
  if (fail.code != NE_NOERROR) {
    exit_status = 1;
    goto END;
  }

  /* We have no explicit objective vector so set lenc = 0; the
    objective vector is stored in row iobj of acol. */
  lenc = 0;
  objadd = 0.;
#ifdef _WIN32
  strcpy_s(prob, (unsigned)_countof(prob), "        ");
#else
  strcpy(prob, "        ");
#endif

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

  /* Illustrate how to pass information to the user-supplied
     function qphx via the comm structure */
  comm.p = 0;

  if (!(iuser = NAG_ALLOC(ncolh + 1 + nnzh, Integer)) ||
    !(ruser = NAG_ALLOC(nnzh, double)))
  {
    printf("Allocation failure\n");
    exit_status = -3;
    goto END;
  }

  if (ncolh > 0) {
    /* Store the nonzeros of H in ruser for use by qphx. */
    for (i = 0; i < nnzh; i++)
      ruser[i] = h[i];
    /* Store iccolh and irowh in iuser for use by qphx. */
    for (i = 0; i < ncolh + 1; i++)
      iuser[i] = iccolh[i];
    for (i = ncolh + 1, j = 0; i < nnzh + ncolh + 1; i++, j++)
      iuser[i] = irowh[j];
    comm.iuser = iuser;
    comm.user = ruser;
  }
  /* nag_opt_sparse_convex_qp_init (e04npc).
   Initialization function for nag_opt_sparse_convex_qp_solve (e04nqc). */
  nag_opt_sparse_convex_qp_init(&state, NAGERR_DEFAULT);

  /* nag_opt_sparse_convex_qp_solve (e04nqc).
     LP or QP problem (suitable for sparse problems). */
  nag_opt_sparse_convex_qp_solve(start, qphx, m, n, ne, nname, lenc,
                                 ncolh, iobj, objadd, pnames[0], acol, inda,
                                 loca, bl, bu, NULL, (const char **) names,
                                 helast, hs, x, pi, rc, &ns, &ninf, &sinf,
                                 &obj, &state, &comm, &fail);

  if (fail.code != NE_NOERROR) {
    printf("nag_opt_sparse_convex_qp_solve (e04nqc) failed.\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print objective function and optimal x* */
  printf("Final objective value = %12.3e\n", obj);
  printf("Optimal X = ");

  for (i = 0; i < n; ++i) {
    printf("%9.2f%s", x[i], i % 7 == 6 || i == n - 1 ? "\n" : " ");
  }

  /* Set the rest of input for e04mwc. Due to iobj > 0, lenc = 0 and c and idxc
     will not be referenced (are optional arguments)
     c - optional
     idxc - optional
     intvar - optional */

  /* pnames */
#ifdef _WIN32
  strcpy_s(pnames[0], 9, "USRPNAME");
#else
  strcpy(pnames[0], "USRPNAME");
#endif
#ifdef _WIN32
  strcpy_s(pnames[1], 9, "OBJ.....");
#else
  strcpy(pnames[1], "OBJ.....");
#endif
#ifdef _WIN32
  strcpy_s(pnames[2], 9, "RHS.....");
#else
  strcpy(pnames[2], "RHS.....");
#endif
#ifdef _WIN32
  strcpy_s(pnames[3], 9, "RANGE...");
#else
  strcpy(pnames[3], "RANGE...");
#endif
#ifdef _WIN32
  strcpy_s(pnames[4], 9, "BOUND...");
#else
  strcpy(pnames[4], "BOUND...");
#endif
  lintvar = 0;
  minmax = -1;

  /* transform char *names[] to (*crname)[9] */
  for (i = 0; i < n + m; i++)
#ifdef _WIN32
    strcpy_s(crname[i], 9, names[i]);
#else
    strcpy(crname[i], names[i]);
#endif
```

```
  /* open data file for writing */
  nag_open_file(fname, 1, &fileid, NAGERR_DEFAULT);

  /* nag_opt_miqp_mps_write (e04mwc).
    writes data for sparse lP, MILP, QP or MIQP problems to a
    file in MPS format. */
  nag_opt_miqp_mps_write(fileid, n, m, lenc, ne, ncolh, nnzh, lintvar,NULL,NULL,
          iobj, acol, inda, loca, bl, bu, pnames, crname, h, irowh,
          iccolh, minmax, NULL, &fail);

  if (fail.code != NE_NOERROR) {
    printf("nag_opt_miqp_mps_write (e04mwc) failed.\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print name of the outfile */
  printf("\nMPS file was written: %s\n", fname);

  /* Close data file */
  nag_close_file(fileid, NAGERR_DEFAULT);

END:

  for (i = 0; i < n + m; i++) {
    NAG_FREE(names[i]);
  }
  NAG_FREE(acol);
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(h);
  NAG_FREE(c);
  NAG_FREE(pi);
  NAG_FREE(rc);
  NAG_FREE(x);
  NAG_FREE(helast);
  NAG_FREE(hs);
  NAG_FREE(loca);
  NAG_FREE(iccolh);
  NAG_FREE(inda);
  NAG_FREE(irowh);
  NAG_FREE(iuser);
  NAG_FREE(ruser);
  NAG_FREE(crname);
  NAG_FREE(names);
  return exit_status;
}

static void NAG_CALL qphx(Integer ncolh, const double x[], double hx[],
                          Integer nstate, Nag_Comm *comm)
{
  /* Function to compute H*x.
     Note: comm->iuser and comm->user contain the following data:
     comm->user[0:nnzh-1] = h[0:nnzh-1]
     comm->iuser[0:ncolh] = iccolh[0:ncolh]
     comm->iuser[ncolh+1:nnzh+ncolh] = irowh[0:nnzh-1] */

  Integer i, end, icol, idx, irow, start;

  for (i = 0; i < ncolh; i++)
    hx[i] = 0.0;
  for (icol = 0; icol < ncolh; icol++) {
    start = comm->iuser[icol];
    end = comm->iuser[icol + 1] - 1;
    for (idx = start - 1; idx < end; idx++) {
      irow = comm->iuser[ncolh + 1 + idx] - 1;
      hx[irow] = hx[irow] + x[icol] * comm->user[idx];
      if (irow != icol)
        hx[icol] = hx[icol] + x[irow] * comm->user[idx];
    }
  }
}
```

10.2 Program Data

```
nag_opt_miqp_mps_write (e04mwc) Example Program Data
 7  8                       : Values of N and M
48  8  7  9  15             : Values of NNZ, IOBJ, NCOLH, NNZH 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

  2.0      1   1    : Sparse matrix H, ordered by increasing column index;
  2.0      2   2    : each row contains H(i), IROWH(i), ICOL (= column index)
  2.0      3   3    : The row indices may be in any order.
  2.0      4   3
  2.0      4   4
  2.0      5   5
  2.0      6   6
  2.0      7   6
  2.0      7   7    : End of matrix H

 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  0.0  0.0  0.0  0.0       : Initial vector X
```

10.3 Program Results

```
nag_opt_miqp_mps_write (e04mwc) Example Program Results

Final objective value =   -1.848e+06
Optimal X =      0.00    349.40    648.85    172.85    407.52    271.36    150.02

MPS file was written: e04mwce.mps
```