nag_opt_sparse_mps_read (e04mzc)

1. Purpose

nag_opt_sparse_mps_read (e04mzc) reads data for a sparse linear programming or quadratic programming problem from a file which is in standard or compatible MPSX input format.

2. Specification

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

void nag_opt_sparse_mps_read(char *mps_file, Integer *n, Integer *m,
                             Integer *nnz, Integer *iobj, double **a,
                             Integer **ha, Integer **ka,
                             double **bl, double **bu, double **xs,
                             Nag_E04_Opt *e04_options, NagError *fail)
```

3. Description

nag_opt_sparse_mps_read reads linear programming (LP) or quadratic programming (QP) problem data from a file which is prepared in standard or compatible MPSX input format and then initializes n (the number of variables), m (the number of general linear constraints), the m by n matrix A, and the vectors l, u and c (stored in row **iobj** of A) for use with nag_opt_sparse_convex_qp (e04nkc), which is designed to solve problems of the form

$$\underset{x \in R^n}{\text{minimize}} \quad c^T x + \tfrac{1}{2} x^T H x \quad \text{subject to} \quad l \leq \left\{ \begin{array}{c} x \\ Ax \end{array} \right\} \leq u. \tag{1}$$

For LP problems, $H = 0$. For QP problems, a function must be provided to nag_opt_sparse_convex_qp (e04nkc) to compute Hx for any given vector x. (This is illustrated in Section 6.) The optional parameter **minimize** may be used to specify whether the objective function is to be minimized or maximized. The document for nag_opt_sparse_convex_qp (e04nkc) should be consulted for further details.

Since, in general, the exact size of the problem defined by an MPSX file may not be known in advance, the arrays returned by nag_opt_sparse_mps_read are all allocated internally.

MPSX Input Format

The MPSX data file may only contain two types of line:

(a) Indicator lines (specifying the type of data which is to follow).

(b) Data lines (specifying the actual data).

The input file must not contain any blank lines. Any characters beyond column 80 are ignored. Indicator lines must not contain leading blank characters (in other words they must begin in column 1). The following displays the order in which the indicator lines must appear in the file:

```
NAME    user-supplied name
ROWS
        data line(s)
COLUMNS
        data line(s)
RHS
        data line(s)
RANGES  (optional)
        data line(s)
BOUNDS  (optional)
        data line(s)
ENDATA
```

The 'user-supplied name' specifies a name for the problem and must occupy columns 15–22. The name can either be blank or up to a maximum of 8 characters.

A data line follows the same fixed format made up of fields defined below. The contents of the fields may have different significance depending upon the section of data in which they appear.

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2–3	5–12	15–22	25–36	40–47	50–61
Contents	Code	Name	Name	Value	Name	Value

The names and codes consist of 'alphanumeric' characters (i.e., a−z, A−Z, $0-9$, +, −, asterisk (*), blank ( ), colon (:), dollar sign ($) or full stop (.) only) and the names must not contain leading blank characters. Values may be entered in several equivalent forms. For example, 1.2345678, 1.2345678e+0, 123.45678e−2 and 12345678E−07 all represent the same number. It is safest to include an explicit decimal point. Note that the lower case 'e' exponential notation is not standard MPSX, and if compatibility with other MPSX readers is required then the upper case notation should be used. The lower case notation is supported by nag_opt_sparse_mps_read since this is the natural notation in a C programming language environment.

It is recommended that numeric values be *right-justified* in the 12-character field, with no trailing blanks. This is to ensure compatibility with other MPSX readers, some of which may, in certain situations, interpret trailing blanks as zeros. This can dramatically affect the interpretation of the value and is relevant if the value contains an exponent, or if it contains neither an exponent nor an explicit decimal point.

Comment lines are allowed in the data file. These must have an asterisk (*) in column 1 and any characters in columns 2–80. In any data line, a dollar sign ($) as the first character in field 3 or 5 indicates that the information from that point through column 80 consists of comments.

Columns outside the six fields must be blank, except for columns 72–80, whose contents are ignored by the routine. These columns may be used to enter a sequence number. A non-blank character outside the predefined six fields and columns 72–80 is considered to be a major error unless it is part of a comment.

ROWS Data Lines

These lines specify row (constraint) names and their inequality types (i.e., =, ≥ or ≤).

Field 1: defines the constraint type as follows (may be in column 2 or column 3):

`N` free row, i.e., no constraint. It may be used to define the objective row.

`G` greater than or equal to (i.e., ≥).

`L` greater than or equal to (i.e., ≤).

`E` exactly equal to (i.e., =).

Field 2: defines the row name.

Row type N stands for 'Not binding', also known as 'Free'. It can be used to define the objective row. The objective row is a free row that specifies the vector c in the linear objective term $c^T x$. It is taken to be the first free row, unless some other free row name is specified by the optional parameter **obj_name** (see Section 7.2). Note that c is assumed to be zero if (for example) the line

`%N%%DUMMYROW`

(where `%` denotes a blank) appears in the ROWS section of the MPSX data file, and the row name `DUMMYROW` is omitted from the COLUMNS section.

COLUMNS Data Lines

These lines specify the names to be assigned to the variables (columns) in the general linear constraint matrix A, and define, in terms of column vectors, the actual values of the corresponding matrix elements.

Field 1: blank (ignored).

Field 2: gives the name of the column associated with the elements specified in the following fields.

Field 3: contains the name of a row.

Field 4: used in conjunction with field 3; contains the value of the matrix element.

Field 5: is optional (may be used like field 3).

Field 6: is optional (may be used like field 4).

Note that only the non-zero elements of A and c need to be specified in the COLUMNS section, as any zero elements of A are removed and any unspecified elements of c are assumed to be zero. In addition, any non-zero elements in the jth column of A must be grouped together before those in the $(j+1)$th column, for $j = 1, 2, \ldots, \mathbf{n}-1$. Non-zero elements within a column may however appear in any order.

RHS Data Lines

This section specifies the right-hand side values of the general linear constraint matrix A (if any). The lines specify the name to be given to the right-hand side (RHS) vector along with the numerical values of the elements of the vector, which may appear in any order. The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RHS name. Only the non-zero elements need be specified. Note that this section may be empty, in which case the RHS vector is assumed to be zero.

RANGES Data Lines (optional)

Ranges are used for constraints of the form $l \le Ax \le u$, where both l and u are finite. The effect of specifying a range r_j for constraint j depends on the type of the constraint (i.e., `G`, `L` or `E`), the sign of r_j, and the bound associated with the constraint in the RHS section. (Recall that this bound is taken to be zero if the constraint has no entry in the RHS section.) The various possibilities may be summarized as follows.

Row Type	Sign of r_j	Bound from RHS	Resultant l_j	Resultant u_j
`G`	+ or −	l_j	l_j	$l_j + \lvert r_j \rvert$
`L`	+ or −	u_j	$u_j - \lvert r_j \rvert$	u_j
`E`	+	l_j	l_j	$l_j + r_j$
`E`	−	u_j	$u_j - \lvert r_j \rvert$	u_j

The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RANGE name.

BOUNDS Data Lines (optional)

These lines specify limits on the values of the variables (l and u in $l \le x \le u$). If the variable is not specified in the bound set then it is automatically assumed to lie between default lower and upper bounds (usually 0 and $+\infty$). (These default bounds may be reset to the values specified by the optional parameters **col_lo_default** and **col_up_default**; see Section 7.2.) Like an RHS column which is given a name, the set of variables in one bound set is also given a name.

Field 1: specifies the type of bound or defines the variable type as follows:

- `LO` lower bound.
- `UP` upper bound.
- `FX` fixed variable.
- `FR` free variable ($-\infty$ to $+\infty$).
- `MI` lower bound is $-\infty$.
- `PL` upper bound is $+\infty$. This is the default variable type.

Field 2: identifies a name for the bound set.

Field 3: identifies the column name of the variable belonging to this set.

Field 4: identifies the value of the bound; this has a numerical value only in association with `LO`, `UP`, `FX` in field 1, otherwise it is blank.

Field 5: is blank and ignored.

Field 6: is blank and ignored.

Note that if RANGES and BOUNDS sections are both present, the RANGES section must appear first.

MPSX and Integer Programming Problems

The MPSX input format allows the specification of *integer programming* (IP) problems in which some or all of the variables are constrained to take integer values within a specified range. nag_opt_sparse_mps_read can read MPSX files defining IP problems in either the 'compatible' or 'standard' formats. However, any integer restrictions are ignored: any variable upon which such restrictions are defined by the file is simply treated as a continuous variable with upper and lower bounds as specified. The facility to read such files is offered to allow users to solve IP problems in their 'relaxed' LP or QP form using nag_opt_sparse_convex_qp (e04nkc). The compatible and standard MPSX forms are described below. Users not interested in this facility may skip the remainder of this section.

In the compatible MPSX format, the type of integer variables are defined in Field 1 of the BOUNDS section, that is:

Field 1: specifies the type of the integer variable as follows:

`BV` 0–1 integer variable (bound value is 1.0).

`UI` general integer variable (bound value is in field 4).

In the standard MPSX format, the integer variables are treated the same as 'ordinary' bounded variables, in the BOUNDS section. Integer markers are, however, introduced in the COLUMNS section to specify the integer variables. The indicator lines for these markers are:

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2–3	5–12	15–22	25–36	40–47	50–61
Contents		*name*	`'MARKER'`		`'INTORG'`	

to mark the beginning of the integer variables and

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2–3	5–12	15–22	25–36	40–47	50–61
Contents		*name*	`'MARKER'`		`'INTEND'`	

to mark the end. That is, any variables between these markers are treated as integer variables. The *name* in Field 2 may be any name different from the preceding and following column names, the other entries in the indicator lines must be exactly as described above (including quotation marks). Note that if the INTEND indicator line is not specified then all columns between the INTORG indicator line and the end of the COLUMNS section are assumed to be integer variables. nag_opt_sparse_mps_read accepts both standard and/or compatible MPSX format as a means of specifying integer variables.

4. Parameters

mps_file

Input: the name of the MPSX data file. If **mps_file** is a null pointer or null string, then the data is assumed to come from `stdin`.

n

Output: the number of columns (variables) specified by the data file.

m

Output: the number of rows specified by the data file. This is the number of general linear constraints in the problem, including the objective row.

nnz

Output: the number of non-zeros in the problem (including the objective row).

iobj

Output: if **iobj** > 0, row **iobj** of A is a free row containing the non-zero coefficients of the vector c (the rows of A are indexed $1, 2, \ldots,$**m**). If **iobj** $= 0$, the coefficients of c are assumed to be zero.

a

Output: the **nnz** non-zero elements of A, ordered by increasing column index.

Sufficient memory is allocated internally by nag_opt_sparse_mps_read and may be freed by the utility function nag_opt_sparse_mps_free (e04myc).

ha

Output: the **nnz** row indices of the non-zero elements of A.

Sufficient memory is allocated internally by nag_opt_sparse_mps_read and may be freed by the utility function nag_opt_sparse_mps_free (e04myc).

ka

Output: the **n** + 1 indices indicating the beginning of each column of A in **a**. More precisely, **ka**$[j-1]$ contains the index in **a** of the start of the jth column, for $j = 1, 2, \ldots,$**n**-1. Note that **ka**$[0] = 0$ and **ka**[**n**] = **nnz**.

Sufficient memory is allocated internally by nag_opt_sparse_mps_read and may be freed by the utility function nag_opt_sparse_mps_free (e04myc).

bl
bu

Output: **bl** and **bu** hold the lower bounds and upper bounds, respectively, for all the variables and constraints, in the following order. The first **n** elements contain the bounds on the variables x and the next **m** elements contain the bounds for the linear objective term $c^T x$ and the general linear constraints Ax (if any). Note that an 'infinite' lower bound is indicated by **bl**$[j-1] = -10^{20}$, an 'infinite' upper bound by **bl**$[j-1] = 10^{20}$, and an equality constraint by **bl**$[j-1]$ = **bu**$[j-1]$. (The lower bound for $c^T x$, stored in **bl**[**n**+**iobj**-1], is set to $-$**options.col_up_default**, and the upper bound, stored in **bl**[**n**+**iobj**-1] is set to **options.col_up_default**; the optional parameter **col_lo_default** has a default value of 10^{20}; see Section 7.)

Sufficient memory is allocated internally by nag_opt_sparse_mps_read and may be freed by the utility function nag_opt_sparse_mps_free (e04myc).

xs

Output: a set of initial values for the **n** variables and **m** constraints in the problem. More precisely, **xs**$[j]$ = min(max(0.0,**bl**$[j]$), **bu**$[j]$), for $j = 0, 1, \ldots,$**m**+**n**-1.

Sufficient memory is allocated internally by nag_opt_sparse_mps_read and may be freed by the utility function nag_opt_sparse_mps_free (e04myc).

options

Input/Output: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_sparse_mps_read. These structure members offer the means of adjusting the parameter values used when reading in the MPSX file and on output will supply further details of the results. A description of the members of **options** is given below in Section 7.2.

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_sparse_mps_read. However, if the optional parameters are not required the NAG defined null pointer, `E04_DEFAULT`, can be used in the function call.

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.

4.1. Description of Printed Output

Results are printed out by default. The level of printed output can be controlled by the user with the structure members **options.list** and **options.output_level** (see Section 7.2). If **list** = **TRUE** then the parameter values to nag_opt_sparse_mps_read are listed, whereas the printout of results is governed by **output_level**. The default print level of **Nag_MPS_Summary** gives the following information if the MPSX file has been read successfully:

(a) the number of lines read.

(b) the number of columns specified by the data. If any of these are specified as integer variables, the number of such variables is also reported. (However, recall that nag_opt_sparse_mps_read will nevertheless regard such variables as continuous variables; see Section 3).

(c) the number of rows specified by the data. The objective row is counted amongst these.

In addition, the names of the problem, the objective row, the RHS set, the RANGES set, and the BOUNDS set read are listed. Unless specified otherwise by the optional parameters **prob_name**, **obj_name**, **rhs_name**, **range_name** and/or **bnd_name** (see Section 7), these names will correspond to the first problem, objective row, etc., encountered in the MPSX file. Where no set was encountered (RANGES and BOUNDS are optional), a 'blank' is output.

Additionally, when **output_level** = **Nag_MPS_List**, each line of the MPSX file is echoed as it is read. This may be useful as a debugging aid.

If **output_level** = **Nag_MPS_NoOutput** then printout will be suppressed; the user can print the information contained in (b) and (c) when nag_opt_sparse_mps_read returns to the calling program.

5. Comments

A list of possible error exits and warnings from nag_opt_sparse_mps_read is given in Section 8.

6. Example 1

To solve the quadratic programming problem

$$\begin{array}{ll} \text{minimize} & c^T x + \tfrac{1}{2} x^T H x \\ \text{subject to} & \begin{array}{rcl} l & \le Ax \le & u, \\ -2 & \le \;\; x \le & 2, \end{array} \end{array}$$

where

$$c = \begin{pmatrix} -4.0 \\ -1.0 \\ -1.0 \\ -1.0 \\ -1.0 \\ -1.0 \\ -1.0 \\ -0.1 \\ -0.3 \end{pmatrix}, \quad H = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 \\ 1 & 2 & 3 & 4 & -2 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad l = \begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix} \quad \text{and} \quad u = \begin{pmatrix} 1.5 \\ 1.5 \\ 4.0 \end{pmatrix}.$$

The optimal solution (to five significant figures) is

$$x^* = (2.0,\ -0.23333,\ -0.26667,\ -0.3,\ -0.1,\ 2.0,\ 2.0,\ -1.7777,\ -0.45555)^T.$$

Three bound constraints and two general linear constraints are active at the solution. Note that, although the Hessian is positive semi-definite, the point x^* is the unique solution.

This example shows the simple use of nag_opt_sparse_mps_read where default values are used for all parameters, in conjunction with nag_opt_sparse_convex_qp (e04nkc). An example showing the use of the optional parameters is given in Section 12. 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. The function to calculate Hx (required by nag_opt_sparse_convex_qp (e04nkc)) is `qphess` for this example.

Note the use of nag_opt_sparse_mps_free (e04myc) in this example to free the memory returned by nag_opt_sparse_mps_read, once the problem has been solved.

The MPSX representation of the problem is given in Section 6.2.

6.1. Program Text

```
/* nag_opt_sparse_mps_read(e04mzc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <nage04.h>

#ifdef NAG_PROTO
static void qphess(Integer ncolh, double x[], double hx[], Nag_Comm *comm);

static void ex1(void);
static void ex2(void);
#else
static void qphess();
static void ex1();
static void ex2();
#endif


main()
{
  /* Two examples are called: ex1() uses the
   * default settings to read an MPSX file and
   * solve the problem, while ex2() uses the
   * options structure to obtain the column
   * and row names.
   */
  Vprintf("e04mzc Example Program Results.\n");
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void ex1(void)
#else
static void ex1()
#endif
{
  Integer iobj, m, n, ncolh, nnz;
  Integer ninf;
  Integer *ha, *ka;
  double *bl, *bu, *a, *xs;
  double obj, sinf;
  static NagError fail;

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

  fail.print = TRUE;

  /* Read the MPSX file */
  e04mzc((char*)0, &n, &m, &nnz, &iobj, &a, &ha, &ka, &bl, &bu, &xs,
         E04_DEFAULT, &fail);

  if (fail.code == NE_NOERROR)
```

```
    {
      /* Solve the problem */
      ncolh = 5;
      e04nkc(n, m, nnz, iobj, ncolh, qphess, a, ha, ka, bl, bu, xs,
             &ninf, &sinf, &obj, E04_DEFAULT, NAGCOMM_NULL, &fail);
    }

  /* Free the memory allocated by e04mzc */
  e04myc(&a, &ha, &ka, &bl, &bu, &xs);

}

#ifdef NAG_PROTO
static void qphess(Integer ncolh, double x[], double hx[], Nag_Comm *comm)
#else
static void qphess(ncolh, x, hx, comm)
     Integer ncolh;
     double x[], hx[];
     Nag_Comm *comm;
#endif
{
  /*  Function to compute H*x. */
  hx[0] = 2.0*x[0] + x[1] + x[2] + x[3] + x[4];
  hx[1] = x[0] + 2.0*x[1] + x[2] + x[3] + x[4];
  hx[2] = x[0] + x[1] + 2.0*x[2] + x[3] + x[4];
  hx[3] = x[0] + x[1] + x[2] + 2.0*x[3] + x[4];
  hx[4] = x[0] + x[1] + x[2] + x[3] + 2.0*x[4];
}  /* qphess */


/* Example 2 */
```

6.2. Program Data

```
* e04mzc Example Program Data
*
* Example 1 MPSX data
*
NAME          QP
ROWS
 L  R1......
 L  R2......
 L  R3......
 N  Free Row
COLUMNS
    X1......  R1......      1.0       R2......         1.0
    X1......  R3......      1.0       Free Row        -4.0
    X2......  R1......      1.0       R2......         2.0
    X2......  R3......     -1.0       Free Row        -1.0
    X3......  R1......      1.0       R2......         3.0
    X3......  R3......      1.0       Free Row        -1.0
    X4......  R1......      1.0       R2......         4.0
    X4......  R3......     -1.0       Free Row        -1.0
    X5......  R1......      1.0       R2......        -2.0
    X5......  R3......      1.0       Free Row        -1.0
    X6......  R1......      1.0       R2......         1.0
    X6......  R3......      1.0       Free Row        -1.0
    X7......  R1......      1.0       R2......         1.0
    X7......  R3......      1.0       Free Row        -1.0
    X8......  R1......      1.0       R2......         1.0
    X8......  R3......      1.0       Free Row        -0.1
    X9......  R1......      4.0       R2......         1.0
    X9......  R3......      1.0       Free Row        -0.3
RHS
    RHS1      R1......      1.5
    RHS1      R2......      1.5
    RHS1      R3......      4.0
RANGES
    RANGE1    R1......      3.5
    RANGE1    R2......      3.5
```

```
    RANGE1    R3......      6.0
BOUNDS
 LO BOUND     X1......     -2.0
 LO BOUND     X2......     -2.0
 LO BOUND     X3......     -2.0
 LO BOUND     X4......     -2.0
 LO BOUND     X5......     -2.0
 LO BOUND     X6......     -2.0
 LO BOUND     X7......     -2.0
 LO BOUND     X8......     -2.0
 LO BOUND     X9......     -2.0
 UP BOUND     X1......      2.0
 UP BOUND     X2......      2.0
 UP BOUND     X3......      2.0
 UP BOUND     X4......      2.0
 UP BOUND     X5......      2.0
 UP BOUND     X6......      2.0
 UP BOUND     X7......      2.0
 UP BOUND     X8......      2.0
 UP BOUND     X9......      2.0
ENDATA
*
```

6.3. Program Results

```
e04mzc Example Program Results.

Example 1: default options used.

Parameters to e04mzc
--------------------

prob_name...............  (first)
obj_name................  (first)    rhs_name................  (first)
range_name..............  (first)    bnd_name................  (first)
col_lo_default.......... 0.00e+00    col_up_default.......... 1.00e+20
ncol_approx.............      500    nrow_approx.............      500
est_density............. 5.00e-02
output_level......Nag_MPS_Summary
outfile.................   stdout


MPS file successfully read.
Number of lines read:       57
Number of columns:           9
Number of rows:              4  (including objective row)

MPS Names Selected:
Problem   QP
Objective Free Row  RHS       RHS1
RANGES    RANGE1    BOUNDS    BOUND

MPS data successfully assigned to problem data.

Parameters to e04nkc
--------------------

Problem type............sparse QP    Number of variables.....        9
Linear constraints......        4    Hessian columns.........        5

prob_name...............
obj_name................             rhs_name................
range_name..............             bnd_name................
crnames.............not supplied

minimize................     TRUE    start................... Nag_Cold
ftol.................... 1.00e-06    reset_ftol..............    10000
fcheck..................       60    factor_freq.............      100
scale.............Nag_ExtraScale    scale_tol............... 9.00e-01
optim_tol............... 1.00e-06    max_iter................       65
crash.............Nag_CrashTwice    crash_tol............... 1.00e-01
```

```
partial_price..........       10    pivot_tol.............. 2.04e-11
max_sb.................        6
inf_bound.............. 1.00e+20    inf_step............... 1.00e+20
lu_factor_tol.......... 1.00e+02    lu_update_tol.......... 1.00e+01
lu_sing_tol............ 2.04e-11    machine precision...... 1.11e-16
print_level.........Nag_Soln_Iter
outfile................   stdout

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

   Itn      Step   Ninf   Sinf/Objective   Norm rg
     0   0.0e+00      0     0.000000e+00   0.0e+00
Itn       0 -- Factorize detected a singular Hessian.
              Rank = 5, Diag, min diag = 0.0e+00, 3.3e-13.
     0   0.0e+00      0     0.000000e+00   6.3e+00
Itn 0 -- Feasible QP solution.
     1   7.5e-01      0    -4.375000e+00   3.5e-01
     2   1.0e+00      0    -4.400000e+00   0.0e+00
     3   1.9e-01      0    -4.700000e+00   1.6e+00
     4   1.0e+00      0    -5.100000e+00   2.2e-16
Itn       5 -- Hessian numerically indefinite.
              Square of diag, min diag =  -1.4e-14,   2.7e-13.
     5   2.2e-01      0    -5.932500e+00   3.7e+00
     6   4.7e-01      0    -6.179688e+00   1.1e+00
     7   1.0e+00      0    -6.267162e+00   2.2e-16
     8   8.1e-03      0    -6.532489e+00   1.4e+00
     9   6.3e-01      0    -7.583236e+00   1.3e-15
    10   1.0e+00      0    -8.067718e+00   8.9e-16
    11   1.0e+00      0    -8.067778e+00   2.2e-16

Variable State        Value   Lower Bound  Upper Bound   Lagr Mult    Residual
X1......    UL  2.00000e+00   -2.0000e+00   2.0000e+00  -8.000e-01   0.000e+00
X2......   SBS -2.33333e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.767e+00
X3......   SBS -2.66667e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.733e+00
X4......    BS -3.00000e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.700e+00
X5......   SBS -1.00000e-01   -2.0000e+00   2.0000e+00  -1.570e-16   1.900e+00
X6......    UL  2.00000e+00   -2.0000e+00   2.0000e+00  -9.000e-01   0.000e+00
X7......    UL  2.00000e+00   -2.0000e+00   2.0000e+00  -9.000e-01   0.000e+00
X8......   SBS -1.77778e+00   -2.0000e+00   2.0000e+00   1.761e-17   2.222e-01
X9......    BS -4.55556e-01   -2.0000e+00   2.0000e+00  -5.551e-17   1.544e+00

Constrnt State        Value   Lower Bound  Upper Bound   Lagr Mult    Residual
R1......    UL  1.50000e+00   -2.0000e+00   1.5000e+00  -6.667e-02   0.000e+00
R2......    UL  1.50000e+00   -2.0000e+00   1.5000e+00  -3.333e-02   0.000e+00
R3......    BS  3.93333e+00   -2.0000e+00   4.0000e+00   0.000e+00  -6.667e-02
Free Row    BS -1.07856e+01          None         None  -1.000e+00  -1.079e+01

Exit after 11 iterations.

Optimal QP solution found.

Final QP objective value =  -8.0677778e+00
```

7. Optional Parameters

A number of optional input and output parameters to nag_opt_sparse_mps_read 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_sparse_mps_read; 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.

7.1. Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag_opt_sparse_mps_read together with their default values where relevant.

`Boolean list`	**TRUE**
`Nag_OutputType output_level`	**Nag_MPS_Summary**
`char outfile[80]`	`stdout`
`char prob_name[9]`	'\0'
`char obj_name[9]`	'\0'
`char rhs_name[9]`	'\0'
`char range_name[9]`	'\0'
`char bnd_name[9]`	'\0'
`double col_lo_default`	0.0
`double col_up_default`	10^{20}
`Integer ncol_approx`	500
`Integer nrow_approx`	500
`double est_density`	0.05
`char **crnames`	size **n**+**m**

7.2. Description of Optional Parameters

list – Boolean Default = **TRUE**

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

output_level – Nag_OutputType Default = **Nag_MPS_Summary**

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

Nag_NoOutput No output.

Nag_MPS_Summary A summary of the dimensions of the problem read and a list of the 'MPSX names' (problem name, objective row name, etc.).

Nag_MPS_List As **Nag_MPS_Summary** but each line of the MPSX file is echoed as it is read. This can be useful for debugging the file.

Constraint: **options.output_level** = **Nag_NoOutput**, **Nag_MPS_Summary** or **Nag_MPS_List**.

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.

prob_name – char[9] Default: **prob_name**[0] = '\0'
obj_name – char[9] Default: **obj_name**[0] = '\0'
rhs_name – char[9] Default: **rhs_name**[0] = '\0'
range_name – char[9] Default: **range_name**[0] = '\0'
bnd_name – char[9] Default: **bnd_name**[0] = '\0'

Input: these options contain the names associated with the MPSX form of the problem. These names must be specified as follows:

prob_name must contain the name of the problem to be read or be blank. The problem name is specified in the NAME indicator line (see Section 3) and if **prob_name** is not blank, then nag_opt_sparse_mps_read will search the MPSX file for the specified problem. If **prob_name** is blank, then the first problem encountered will be read.

obj_name must contain the name of the objective row or be blank (in which case the first objective free row is used).

rhs_name must contain the name of the RHS set to be used or be blank (in which case the first RHS set is used).

range_name must contain the name of the RANGES set to be used or be blank (in which case the first RANGES set, if any, is used).

bnd_name must contain the name of the BOUNDS set to be used or be blank (in which case the first BOUNDS set, if any, is used).

Constraints: the names must be valid MPSX names, i.e., they must consist only of the 'alphanumeric' characters as specified in Section 3 and must not contain leading blank characters.
Output: the members contain the appropriate names as read from the MPSX file. Any names specified on input which are not found in the MPSX file are unchanged on exit but will give rise to an error exit from nag_opt_sparse_mps_read (see Section 8).

If the MPSX file is successfully read, the **options** structure can be passed on to nag_opt_sparse_convex_qp (e04nkc), which will solve the problem specified by the file and which can make use of these structure members in its solution output.

col_lo_default – double Default = 0.0

Input: the default lower bound to be used for the variables in the problem when none is specified in the BOUNDS section of the MPSX data file.

col_up_default – double Default = 10^{20}

Input: the default upper bound to be used for the variables in the problem when none is specified in the BOUNDS section of the MPSX data file.
Constraint: **options.col_up_default** $\geq$ **options.col_lo_default**.

ncol_approx – Integer Default = 500
nrow_approx – Integer Default = 500

Input: an estimate of the number of columns and rows in the problem. nag_opt_sparse_mps_read is designed so that the problem size does not have to be known in advance, and allocates memory according to the data contained in the MPSX file. However, for very large problems, an advance estimate of the problem size might allow slightly more efficient memory usage to be achieved. See also the description of optional parameter **est_density**.
Constraints:

options.ncol_approx > 0,

options.nrow_approx > 0.

est_density – double Default = 0.05

Input: an estimate of the density of the nonzeros in the sparse matrix A, i.e., an estimate of **nnz** / (**n** $\times$ **m**). As with the optional parameters **ncol_approx** and **nrow_approx**, if this is known to be significantly larger or smaller than the default, then the user should specify an appropriate value to aid nag_opt_sparse_mps_read in its memory management.
Constraint: **options.est_density** > 0.0.

crnames – char $**$ Default memory **n**+**m** array of char $*$

Output: the MPSX names of all the variables and constraints in the problem in the following order. **crnames**[$j-1$] contains the name of the jth column, for $j = 1, 2, \ldots,$**n**. **crnames**[**n**+ i -1] contains the name of the ith row, for $i = 1, 2, \ldots,$**m**. Each name is 8 characters long, and includes any trailing blank characters which appear in the appropriate name field of the MPSX file.

Sufficient memory to hold the names is allocated internally by nag_opt_sparse_mps_read. The memory freeing function nag_opt_free (e04xzc) should be used to free this memory. Users should **not** use the standard C function `free()` for this purpose.

If, on return from nag_opt_sparse_mps_read, nag_opt_sparse_convex_qp (e04nkc) is called with **options** as an argument, and the memory pointed to by **crnames** has not been freed, nag_opt_sparse_convex_qp (e04nkc) will use the row and column names stored in **crnames** in its solution output.

8. Error Indications and Warnings

NE_NULL_ARGUMENT

Argument **n** is a null pointer. It should contain the address of a variable of type Integer.
Argument **m** is a null pointer. It should contain the address of a variable of type Integer.
Argument **nnz** is a null pointer. It should contain the address of a variable of type Integer.
Argument **iobj** is a null pointer. It should contain the address of a variable of type Integer.
Argument **bl** is a null pointer. It should contain the address of a variable of type double $*$.
Argument **bu** is a null pointer. It should contain the address of a variable of type double $*$.
Argument **a** is a null pointer. It should contain the address of a variable of type double $*$.
Argument **ha** is a null pointer. It should contain the address of a variable of type Integer $*$.
Argument **ka** is a null pointer. It should contain the address of a variable of type Integer $*$.
Argument **xs** is a null pointer. It should contain the address of a variable of type double $*$.

NE_OPT_NOT_INIT

Options structure not initialized.

NE_BAD_PARAM

On entry, parameter **options.prob_name** had an illegal value.
On entry, parameter **options.obj_name** had an illegal value.
On entry, parameter **options.rhs_name** had an illegal value.
On entry, parameter **options.range_name** had an illegal value.
On entry, parameter **options.bnd_name** had an illegal value.
On entry, parameter **options.output_level** had an illegal value.

NE_2_REAL_EE_OPT_ARG_CONS

On entry, **options.col_lo_default** $= \langle value \rangle$ while **options.col_up_default** $= \langle value \rangle$.
Constraint: **options.col_lo_default** $\leq$ **options.col_up_default**.

NE_INT_OPT_ARG_LT

On entry, **options.nrow_approx** $= \langle value \rangle$. Constraint: **nrow_approx** ≥ 1
On entry, **options.ncol_approx** $= \langle value \rangle$. Constraint: **ncol_approx** ≥ 1

NE_INVALID_REAL_RANGE_F

Value $\langle value \rangle$ given to **options.est_density** is not valid. Correct range is **est_density** > 0.0.

NE_NAMES_NOT_NAG_MEM

options.crnames is not null but does not point to memory allocated by an earlier call to this function. This function does not accept user-allocated memory assigned to **crnames**.

NE_MPS_PROB_NOT_FOUND

The specified problem has not been found in the MPSX file.

NE_MPS_ILLEGAL_DATA_LINE

An illegal data line has been read from the MPSX file. This is neither a comment nor a legal data line.
Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_NAME

An illegal row or column name has been detected. Names must contain only alphanumeric characters with no leading blanks.
Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_NUMBER

Number expected but value could not be read. Check numerical fields.
Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_SETNAME

An illegal name has been detected in field 2 of the RHS, RANGES or BOUNDS section. Names must contain only alphanumeric characters with no leading blanks.
Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_BND_TYPE
An invalid bound type appears in the BOUNDS section. Expect: LO, UP, FX, FR, MI, PL, BV or UI.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_INVALID_BND_VAL
Invalid numeric field in bound data. Value expected for types: LO, UP, FX, UI. Blank field expected for types: FR, MI, PL, BV.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_INVALID_INDICATOR
Unknown, unexpected or invalid indicator line read. Expect: NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS or ENDATA, starting in column 1 of file, and in that order. RANGES and/or BOUNDS may be omitted.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_INVALID_INTORG_INTEND
An INTORG or INTEND marker is not correctly specified or is unexpected (e.g., INTEND has no matching INTORG).
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_INVALID_ROW_TYPE
An invalid row type appears in the ROWS section. Expect: N, G, L or E.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_NO_COLS
There were no columns specified in the COLUMNS section.
Last MPSX line read (⟨value⟩): ⟨*string*⟩.

NE_MPS_NO_NEWLINE
New line expected but not found.
Last MPSX line read (⟨value⟩): ⟨*string*⟩.

NE_MPS_NO_OBJ
The objective row was not found. There must be at least one row of type N in the ROWS section and, if an objective name was specified, there must be a type N row with this name.
Last MPSX line read (⟨value⟩): ⟨*string*⟩.

NE_MPS_NO_ROWS
There were no rows specified in the ROWS section.
Last MPSX line read (⟨value⟩): ⟨*string*⟩.

NE_MPS_REPEAT_ROW
A row has been specified more than once.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_RHS_RANGE_BND_NOT_FOUND
The name of the RHS, RANGES or BOUNDS set to be used was not found in the file.

NE_MPS_ENDATA_NOT_FOUND
The file does not contain an ENDATA indicator.

NE_MPS_SPLIT_COL
Column data is not contiguous. All entries for a given column must appear together in the COLUMNS section.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_UNKNOWN_COLNAME
An unknown column name appears in the BOUNDS section. All the column names must be specified in the COLUMNS section.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_MPS_UNKNOWN_ROWNAME
An unknown row name appears in the ⟨*string*⟩ section. All the row names must be specified in the ROWS section.
Error at MPSX line ⟨value⟩: ⟨*string*⟩.

NE_ALLOC_FAIL
Memory allocation failed.

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*⟩.

NE_NOT_READ_FILE
Cannot open file ⟨*string*⟩ for reading.

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 consult NAG for assistance.

9. Further Comments

None.

10. References

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

11. See Also

nag_opt_sparse_mps_free (e04myc)
nag_opt_sparse_convex_qp (e04nkc)
nag_opt_init (e04xxc)
nag_opt_read (e04xyc)
nag_opt_free (e04xzc)

12. Example 2

This example solves the same problem as Example 1 but illustrates the use of the **options** structure. Although the problem is the same, it is defined by a slightly modified MPSX file. The same `qphess` function is used as in Example 1.

The options structure is initialized by a call to nag_opt_init (e04xxc) and two of the optional parameters are set: **prob_name** is set to `"..QP 2.."` so that nag_opt_sparse_mps_read will attempt to read a problem of this name; and **obj_name** is set to `"..COST.."`. The MPSX file (see Section 12.2) contains an additional free row, named `"FREE ROW"`. Since this is the first free row in the ROWS section of the MPSX file, by default it would be read as the objective row. However, since **obj_name** is specified, nag_opt_sparse_mps_read takes the second free row (`"..COST.."`) as the objective row.

nag_opt_sparse_mps_read is called to read the MPSX file, and this is followed by a call to nag_opt_sparse_convex_qp (e04nkc) to solve the problem. As the **options** structure is passed as an argument, the row and column names read from the file are stored in **options.crnames** and used in the solution output (see Section 12.3).

Finally, nag_opt_sparse_mps_free (e04myc) is called to free the problem arrays, and nag_opt_free (e04xzc) is called to free the memory in **options**.

12.1. Program Text

```
static void ex2(void)
#else
static void ex2()
#endif
{
```

```
  Integer iobj, m, n, ncolh, nnz;
  Integer ninf;
  Integer *ha, *ka;
  double *bl, *bu, *a, *xs;
  double obj, sinf;
  Nag_E04_Opt options;
  static NagError fail, fail2;

  Vprintf("\nExample 2: use of options structure.\n");

  fail.print = TRUE;

  /* Initialize the options structure and read MPSX data */
  e04xxc(&options);
  Vstrcpy(options.prob_name, "..QP 2..");
  Vstrcpy(options.obj_name, "..COST..");
  e04mzc((char*)0, &n, &m, &nnz, &iobj, &a, &ha, &ka, &bl, &bu, &xs,
         &options, &fail);

  /* Column and row names are now available via options */

  if (fail.code == NE_NOERROR)
    {
      ncolh = 5;
      e04nkc(n, m, nnz, iobj, ncolh, qphess, a, ha, ka, bl, bu, xs,
             &ninf, &sinf, &obj, &options, NAGCOMM_NULL, &fail);
    }

  /* Free memory returned by e04mzc */
  e04myc(&a, &ha, &ka, &bl, &bu, &xs);

  /* Free memory in options (including column & row names) */
  fail2.print = TRUE;
  e04xzc(&options, "all", &fail2);

}
```

12.2. Program Data

```
 L  ..ROW1..
 L  ..ROW2..
 L  ..ROW3..
 N  FREE ROW
 N  ..COST..
COLUMNS
    ...X1...  ..ROW1..      1.0       ..ROW2..        1.0
    ...X1...  ..ROW3..      1.0       ..COST..       -4.0
    ...X2...  ..ROW1..      1.0       ..ROW2..        2.0
    ...X2...  ..ROW3..     -1.0       ..COST..       -1.0
    ...X3...  ..ROW1..      1.0       ..ROW2..        3.0
    ...X3...  ..ROW3..      1.0       ..COST..       -1.0
    ...X4...  ..ROW1..      1.0       ..ROW2..        4.0
    ...X4...  ..ROW3..     -1.0       ..COST..       -1.0
    ...X5...  ..ROW1..      1.0       ..ROW2..       -2.0
    ...X5...  ..ROW3..      1.0       ..COST..       -1.0
    ...X6...  ..ROW1..      1.0       ..ROW2..        1.0
    ...X6...  ..ROW3..      1.0       ..COST..       -1.0
    ...X7...  ..ROW1..      1.0       ..ROW2..        1.0
    ...X7...  ..ROW3..      1.0       ..COST..       -1.0
    ...X8...  ..ROW1..      1.0       ..ROW2..        1.0
    ...X8...  ..ROW3..      1.0       ..COST..       -0.1
    ...X9...  ..ROW1..      4.0       ..ROW2..        1.0
    ...X9...  ..ROW3..      1.0       ..COST..       -0.3
RHS
    RHS1      ..ROW1..      1.5
    RHS1      ..ROW2..      1.5
    RHS1      ..ROW3..      4.0
RANGES
    RANGE1    ..ROW1..      3.5
    RANGE1    ..ROW2..      3.5
    RANGE1    ..ROW3..      6.0
```

```
BOUNDS
 LO BOUND     ...X1...     -2.0
 LO BOUND     ...X2...     -2.0
 LO BOUND     ...X3...     -2.0
 LO BOUND     ...X4...     -2.0
 LO BOUND     ...X5...     -2.0
 LO BOUND     ...X6...     -2.0
 LO BOUND     ...X7...     -2.0
 LO BOUND     ...X8...     -2.0
 LO BOUND     ...X9...     -2.0
 UP BOUND     ...X1...      2.0
 UP BOUND     ...X2...      2.0
 UP BOUND     ...X3...      2.0
 UP BOUND     ...X4...      2.0
 UP BOUND     ...X5...      2.0
 UP BOUND     ...X6...      2.0
 UP BOUND     ...X7...      2.0
 UP BOUND     ...X8...      2.0
 UP BOUND     ...X9...      2.0
ENDATA
```

12.3. Program Results

```
Parameters to e04mzc
--------------------

prob_name............... ..QP 2..
obj_name................ ..COST..    rhs_name................  (first)
range_name..............  (first)    bnd_name................  (first)
col_lo_default.......... 0.00e+00    col_up_default.......... 1.00e+20
ncol_approx............       500    nrow_approx............       500
est_density............. 5.00e-02
output_level......Nag_MPS_Summary
outfile................    stdout


MPS file successfully read.
Number of lines read:       57
Number of columns:           9
Number of rows:              5  (including objective row)

MPS Names Selected:
Problem   ..QP 2..
Objective ..COST..  RHS       RHS1
RANGES    RANGE1    BOUNDS    BOUND

MPS data successfully assigned to problem data.

Parameters to e04nkc
--------------------

Problem type............sparse QP    Number of variables.....         9
Linear constraints......        5    Hessian columns.........         5

prob_name............... ..QP 2..
obj_name................ ..COST..    rhs_name................ RHS1
range_name.............. RANGE1      bnd_name................ BOUND
crnames................. supplied

minimize...............      TRUE    start................... Nag_Cold
ftol.................... 1.00e-06    reset_ftol.............     10000
fcheck.................        60    factor_freq............       100
scale.............Nag_ExtraScale     scale_tol............... 9.00e-01
optim_tol............... 1.00e-06    max_iter...............        70
crash.............Nag_CrashTwice     crash_tol............... 1.00e-01
partial_price..........        10    pivot_tol............... 2.04e-11
max_sb.................         6
inf_bound............... 1.00e+20    inf_step................ 1.00e+20
lu_factor_tol........... 1.00e+02    lu_update_tol........... 1.00e+01
lu_sing_tol............. 2.04e-11    machine precision....... 1.11e-16
```

```
print_level.........Nag_Soln_Iter
outfile.................   stdout

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

   Itn      Step   Ninf   Sinf/Objective   Norm rg
     0   0.0e+00      0     0.000000e+00   0.0e+00
Itn       0 -- Factorize detected a singular Hessian.
               Rank = 5, Diag, min diag = 0.0e+00, 3.3e-13.
     0   0.0e+00      0     0.000000e+00   6.3e+00
Itn 0 -- Feasible QP solution.
     1   7.5e-01      0    -4.375000e+00   3.5e-01
     2   1.0e+00      0    -4.400000e+00   0.0e+00
     3   1.9e-01      0    -4.700000e+00   1.6e+00
     4   1.0e+00      0    -5.100000e+00   2.2e-16
Itn       5 -- Hessian numerically indefinite.
               Square of diag, min diag =  -1.4e-14,   2.7e-13.
     5   2.2e-01      0    -5.932500e+00   3.7e+00
     6   4.7e-01      0    -6.179688e+00   1.1e+00
     7   1.0e+00      0    -6.267162e+00   2.2e-16
     8   8.1e-03      0    -6.532489e+00   1.4e+00
     9   6.3e-01      0    -7.583236e+00   1.3e-15
    10   1.0e+00      0    -8.067718e+00   8.9e-16
    11   1.0e+00      0    -8.067778e+00   2.2e-16

Variable State          Value   Lower Bound  Upper Bound   Lagr Mult    Residual
...X1...    UL   2.00000e+00   -2.0000e+00   2.0000e+00  -8.000e-01   0.000e+00
...X2...   SBS  -2.33333e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.767e+00
...X3...   SBS  -2.66667e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.733e+00
...X4...    BS  -3.00000e-01   -2.0000e+00   2.0000e+00   0.000e+00   1.700e+00
...X5...   SBS  -1.00000e-01   -2.0000e+00   2.0000e+00  -1.570e-16   1.900e+00
...X6...    UL   2.00000e+00   -2.0000e+00   2.0000e+00  -9.000e-01   0.000e+00
...X7...    UL   2.00000e+00   -2.0000e+00   2.0000e+00  -9.000e-01   0.000e+00
...X8...   SBS  -1.77778e+00   -2.0000e+00   2.0000e+00   1.761e-17   2.222e-01
...X9...    BS  -4.55556e-01   -2.0000e+00   2.0000e+00  -5.551e-17   1.544e+00

Constrnt State          Value   Lower Bound  Upper Bound   Lagr Mult    Residual
..ROW1..    UL   1.50000e+00   -2.0000e+00   1.5000e+00  -6.667e-02   0.000e+00
..ROW2..    UL   1.50000e+00   -2.0000e+00   1.5000e+00  -3.333e-02   0.000e+00
..ROW3..    BS   3.93333e+00   -2.0000e+00   4.0000e+00   0.000e+00  -6.667e-02
FREE ROW    BS   0.00000e+00          None         None   0.000e+00   0.000e+00
..COST..    BS  -1.07856e+01          None         None  -1.000e+00  -1.079e+01

Exit after 11 iterations.

Optimal QP solution found.

Final QP objective value =  -8.0677778e+00
```