NAG Library Function Document

nag_opt_miqp_mps_read (e04mxc)

1 Purpose

nag_opt_miqp_mps_read (e04mxc) reads data for sparse linear programming, mixed integer linear programming, quadratic programming or mixed integer quadratic programming problems from an external file which is in standard or compatible MPS input format.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_miqp_mps_read (Nag_FileID fileid, Integer maxn, Integer maxm,
     Integer maxnnz, Integer maxncolh, Integer maxnnzh, Integer maxlintvar,
     Integer mpslst, Integer *n, Integer *m, Integer *nnz, Integer *ncolh,
     Integer *nnzh, Integer *lintvar, Integer *iobj, double a[],
     Integer irowa[], Integer iccola[], double bl[], double bu[],
     char pnames[][9], Integer *nname, char crname[][9], double h[],
     Integer irowh[], Integer iccolh[], Integer *minmax, Integer intvar[],
     NagError *fail)
```

3 Description

nag_opt_miqp_mps_read (e04mxc) reads data for linear programming (LP) or quadratic programming (QP) problems (or their mixed integer variants) from an external file which is prepared in standard or compatible MPS (see IBM (1971)) input format. It then initializes n (the number of variables), m (the number of general linear constraints), the m by n matrix A, the vectors l, u, c (stored in row **iobj** of A) and the n by n Hessian matrix H for use with nag_opt_sparse_convex_qp_solve (e04nqc). This function is designed to solve problems of the form

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

3.1 MPS input format

The input file of data may only contain two types of lines:

1. Indicator lines (specifying the type of data which is to follow).
2. Data lines (specifying the actual data).

A *section* is a combination of an indicator line and its corresponding data line(s). 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	(optional)
OBJSENSE		(optional)
	data line	
OBJNAME		(optional)
	data line	
ROWS		
	data line(s)	
COLUMNS		
	data line(s)	
RHS		
	data line(s)	
RANGES		(optional)
	data line(s)	
BOUNDS		(optional)
	data line(s)	
QUADOBJ		(optional)
	data line(s)	
ENDATA		

A data line follows a fixed format, being made up of fields as 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

Each name and code must consist of 'printable' characters only; names and codes supplied must match the case used in the following descriptions. Values are read using a field width of 12. This allows values to 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.

Lines with an asterisk (∗) in column 1 will be considered comment lines and will be ignored by the function.

Columns outside the six fields must be blank, except for columns 72–80, whose contents are ignored by the function. A non-blank character outside the predefined six fields and columns 72–80 is considered to be a major error (**fail.code** = NE_MPS_ILLEGAL_DATA_LINE; see Section 6), unless it is part of a comment.

3.1.1 NAME Section (optional)

The NAME section is the only section where the data must be on the same line as the indicator. The 'user-supplied name' must be in field 3 but may be blank.

Field	**Required**	**Description**
3	No	Name of the problem

3.1.2 OBJSENSE Section (optional)

The data line in this section can be used to specify the sense of the objective function. If this section is present it must contain only one data line. If the section is missing or empty, minimization is assumed.

Field	**Required**	**Description**
2	No	Sense of the objective function

Field 2 may contain either `MIN`, `MAX`, `MINIMIZE` or `MAXIMIZE`.

3.1.3 OBJNAME Section (optional)

The data line in this section can be used to specify the name of a free row (see Section 3.1.4) that should be used as the objective function. If this section is present it must contain only one data line. If the section is missing or is empty, the first free row will be chosen instead. Alternatively, OBJNAME can be overridden by setting nonempty **pnames**[1] (see Section 5).

Field	Required	Description
2	No	Row name to be used as the objective function

Field 2 must contain a valid row name.

3.1.4 ROWS Section

The data lines in this section specify unique row (constraint) names and their inequality types (i.e., unconstrained, $=$, $\geq$ or $\leq$).

Field	Required	Description
1	Yes	Inequality key
2	Yes	Row name

The inequality key specifies each row's type. It must be `E`, `G`, `L` or `N` and can be in either column 2 or 3.

Inequality Key	Description	l	u
`N`	Free row	$-\infty$	∞
`G`	Greater than or equal to	finite	∞
`L`	Less than or equal to	$-\infty$	finite
`E`	Equal to	finite	l

Row type `N` stands for 'Not binding'. 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 $c^{\mathrm{T}}x$. If there is more than one free row, the first free row is chosen, unless another free row name is specified by OBJNAME (see Section 3.1.3) or **pnames**[1] (see Section 5). Note that c is assumed to be zero if either the chosen row does not appear in the COLUMNS section (i.e., has no nonzero elements) or there are no free rows defined in the ROWS section.

3.1.5 COLUMNS Section

Data lines in this section 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	Required	Description
2	Yes	Column name
3	Yes	Row name
4	Yes	Value
5	No	Row name
6	No	Value

Each data line in the COLUMNS section defines the nonzero elements of A or c. Any elements of A or c that are undefined are assumed to be zero. Nonzero elements of A must be grouped by column, that is to say that all of the nonzero elements in the jth column of A must be specified before those in the $j+1$th column, for $j = 1, 2, \ldots, n-1$. Rows may appear in any order within the column.

3.1.5.1 Integer Markers

For backward compatibility nag_opt_miqp_mps_read (e04mxc) allows you to define the integer variables within the COLUMNS section using integer markers, although this is not recommended as markers can be treated differently by different MPS readers; you should instead define any integer variables in the BOUNDS section (see below). Each marker line must have the following format:

Field	Required	Description
2	No	Marker ID
3	Yes	Marker tag
5	Yes	Marker type

The marker tag must be 'MARKER'. The marker type must be 'INTORG' to start reading integer variables and 'INTEND' to finish reading integer variables. This implies that a row cannot be named 'MARKER', 'INTORG' or 'INTEND'. Please note that both marker tag and marker type comprise of 8 characters as a ' is the mandatory first and last character in the string. You may wish to have several integer marker sections within the COLUMNS section, in which case each marker section must begin with an 'INTORG' marker and end with an 'INTEND' marker and there should not be another marker between them.

Field 2 is ignored by nag_opt_miqp_mps_read (e04mxc). When an integer variable is declared it will keep its default bounds unless they are changed in the BOUNDS section. This may vary between different MPS readers.

3.1.6 RHS Section

This section specifies the right-hand side values (if any) of the general linear constraint matrix A.

Field	Required	Description
2	Yes	RHS name
3	Yes	Row name
4	Yes	Value
5	No	Row name
6	No	Value

The MPS file may contain several RHS sets distinguished by RHS name. If an RHS name is defined in **pnames**[2] (see Section 5) then nag_opt_miqp_mps_read (e04mxc) will read in only that RHS vector, otherwise the first RHS set will be used.

Only the nonzero RHS elements need to be specified. Note that if an RHS is given to the objective function it will be ignored by nag_opt_miqp_mps_read (e04mxc). An RHS given to the objective function is dealt with differently by different MPS readers, therefore it is safer to not define an RHS of the objective function in your MPS file. Note that this section may be empty, in which case the RHS vector is assumed to be zero.

3.1.7 RANGES Section (optional)

Ranges are used to modify the interpretation of constraints defined in the ROWS section (see Section 3.1.4) to the form $l \leq Ax \leq u$, where both l and u are finite. The range of the constraint is $r = u - l$.

Field	Required	Description
2	Yes	Range name
3	Yes	Row name
4	Yes	Value
5	No	Row name
6	No	Value

The range of each constraint implies an upper and lower bound dependent on the inequality key of each constraint, on the RHS b of the constraint (as defined in the RHS section), and on the range r.

Inequality Key	Sign of r	l	u
E	$+$	b	$b + r$
E	$-$	$b + r$	b
G	$+/-$	b	$b + \lvert r \rvert$
L	$+/-$	$b - \lvert r \rvert$	b
N	$+/-$	$-\infty$	$+\infty$

If a range name is defined in **pnames**[3] (see Section 5) then the function will read in only the range set of that name, otherwise the first set will be used.

3.1.8 BOUNDS Section (optional)

These lines specify limits on the values of the variables (the quantities l and u in $l \leq x \leq u$). If a variable is not specified in the bound set then it is automatically assumed to lie between 0 and $+\infty$.

Field	Required	Description
1	Yes	Bound type identifier
2	Yes	Bound name
3	Yes	Column name
4	Yes/No	Value

Note: field 4 is required only if the bound type identifier is one of `UP`, `LO`, `FX`, `UI` or `LI` in which case it gives the value k below. If the bound type identifier is `FR`, `MI`, `PL` or `BV`, field 4 is ignored and it is recommended to leave it blank.

The table below describes the acceptable bound type identifiers and how each determines the variables' bounds.

Bound Type Identifier	*l*	*u*	Integer Variable?
`UP`	unchanged	k	No
`LO`	k	unchanged	No
`FX`	k	k	No
`FR`	$-\infty$	∞	No
`MI`	$-\infty$	unchanged	No
`PL`	unchanged	∞	No
`BV`	0	1	Yes
`UI`	unchanged	k	Yes
`LI`	k	unchanged	Yes

If a bound name is defined in **pnames**[4] (see Section 5) then the function will read in only the bound set of that name, otherwise the first set will be used.

3.1.9 QUADOBJ Section (optional)

The QUADOBJ section defines nonzero elements of the upper or lower triangle of the Hessian matrix H.

Field	Required	Description
2	Yes	Column name (HColumn Index)
3	Yes	Column name (HRow Index)
4	Yes	Value
5	No	Column name (HRow Index)
6	No	Value

Each data line in the QUADOBJ section defines one (or optionally two) nonzero elements H_{ij} of the matrix H. Each element H_{ij} is given as a triplet of row index i, column index j and a value. The column names (as defined in the COLUMNS section) are used to link the names of the variables and the indices i and j. More precisely, the matrix H on output will have a nonzero element

$$H_{ij} = \text{Value}$$

where index j belongs to HColumn Index and index i to one of the HRow Indices such that

crname$[j-1]$ = Column name (HColumn Index) and

crname$[i-1]$ = Column name (HRow Index).

It is only necessary to define either the upper or lower triangle of the H matrix; either will suffice. Any elements that have been defined in the upper triangle of the matrix will be moved to the lower triangle of the matrix, then any repeated nonzeros will be summed.

Note: it is much more efficient for nag_opt_sparse_convex_qp_solve (e04nqc) to have the H matrix defined by the first **ncolh** column names. If the nonzeros of H are defined by any columns that are not in the first **ncolh** of **n** then nag_opt_miqp_mps_read (e04mxc) will rearrange the matrices A and H so that they are.

3.2 Query Mode

nag_opt_miqp_mps_read (e04mxc) offers a 'query mode' to quickly give upper estimates on the sizes of user arrays. In this mode any expensive checks of the data and of the file format are skipped, providing a prompt count of the number of variables, constraints and matrix nonzeros. This might be useful in the common case where the size of the problem is not known in advance.

You may activate query mode by setting any of the following: **maxn** < 1, **maxm** < 1, **maxnnz** < 1, **maxncolh** < 0 or **maxnnzh** < 0. If no major formatting error is detected in the data file, **fail.code** $=$ NE_NOERROR is returned and the upper estimates are given as stated in Table 1. Alternatively, the function switches to query mode while the file is being read if it is discovered that the provided space is insufficient (that is, if **n** $>$ **maxn**, **m** $>$ **maxm**, **nnz** $>$ **maxnnz**, **ncolh** $>$ **maxncolh**, **nnzh** $>$ **maxnnzh** or **lintvar** $>$ **maxlintvar**). In this case **fail.code** $=$ NE_INT_MAX is returned.

Argument Name	Upper Estimate for
n	**maxn**
m	**maxm**
nnz	**maxnnz**
ncolh	**maxncolh**
nnzh	**maxnnzh**
lintvar	**maxlintvar**

Table 1

The recommended practice is shown in Section 10, where the function is invoked twice. The first call queries the array lengths required, after which the data arrays are allocated to be of these sizes. The second call reads the data using the sufficiently-sized arrays.

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 MPSX data file to be read as returned by a call to nag_open_file (x04acc).

Constraint: **fileid** ≥ 0.

2: **maxn** – Integer *Input*

On entry: an upper limit for the number of variables in the problem.

If **maxn** < 1, nag_opt_miqp_mps_read (e04mxc) will start in query mode (see Section 3.2).

3: **maxm** – Integer *Input*

On entry: an upper limit for the number of general linear constraints (including the objective row) in the problem.

If **maxm** < 1, nag_opt_miqp_mps_read (e04mxc) will start in query mode (see Section 3.2).

4: **maxnnz** – Integer *Input*

On entry: an upper limit for the number of nonzeros (including the objective row) in the problem.

If **maxnnz** < 1, nag_opt_miqp_mps_read (e04mxc) will start in query mode (see Section 3.2).

5: **maxncolh** – Integer *Input*

On entry: an upper limit for the dimension of the matrix H.

If **maxncolh** < 0, nag_opt_miqp_mps_read (e04mxc) will start in query mode (see Section 3.2).

6: **maxnnzh** – Integer *Input*

On entry: an upper limit for the number of nonzeros of the matrix H.

If **maxnnzh** < 0, nag_opt_miqp_mps_read (e04mxc) will start in query mode (see Section 3.2).

7: **maxlintvar** – Integer *Input*

On entry: if **maxlintvar** ≥ 0, an upper limit for the number of integer variables.

If **maxlintvar** < 0, nag_opt_miqp_mps_read (e04mxc) will treat all integer variables in the file as continuous variables.

8: **mpslst** – Integer *Input*

On entry: if **mpslst** $\neq 0$, summary messages are sent to `stdout` as nag_opt_miqp_mps_read (e04mxc) reads through the data file. This can be useful for debugging the file. If **mpslst** $= 0$, then no summary is produced.

9: **n** – Integer * *Output*

On exit: if nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** $=$ NE_INT_MAX, an upper estimate of the number of variables of the problem. Otherwise, n, the actual number of variables in the problem.

10: **m** – Integer * *Output*

On exit: if nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** $=$ NE_INT_MAX, an upper estimate of the number of general linear constraints in the problem (including the objective row). Otherwise m, the actual number of general linear constaints of the problem.

11: **nnz** – Integer * *Output*

On exit: if nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** $=$ NE_INT_MAX, an upper estimate of the number of nonzeros in the problem (including the objective row). Otherwise the actual number of nonzeros in the problem (including the objective row).

12: **ncolh** – Integer * *Output*

On exit: if nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** $=$ NE_INT_MAX, an upper estimate of the value of **ncolh** required by nag_opt_sparse_convex_qp_solve (e04nqc). In this context **ncolh** is the number of leading nonzero columns of the Hessian matrix H. Otherwise, the actual dimension of the matrix H.

13: **nnzh** – Integer * *Output*

On exit: if nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** $=$ NE_INT_MAX, an upper estimate of the number of nonzeros of the matrix H. Otherwise, the actual number of nonzeros of the matrix H.

14: **lintvar** – Integer * *Output*

On exit: if on entry **maxlintvar** < 0, all integer variables are treated as continuous and **lintvar** $= -1$.

If nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** = NE_INT_MAX, an upper estimate of the number of integer variables of the problem. Otherwise, the actual number of integer variables of the problem.

15: **iobj** – Integer * *Output*

On exit: if **iobj** > 0, row **iobj** of A is a free row containing the nonzero coefficients of the vector c.

If **iobj** $= 0$, the coefficients of c are assumed to be zero.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2) **iobj** is not referenced and may be **NULL**.

16: **a**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **a** must be at least **maxnnz** when **maxnnz** > 0.

On exit: the nonzero elements of A, ordered by increasing column index.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **a** is not referenced and may be **NULL**.

17: **irowa**$[dim]$ – Integer *Output*

Note: the dimension, *dim*, of the array **irowa** must be at least **maxnnz** when **maxnnz** > 0.

On exit: the row indices of the nonzero elements stored in **a**.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **irowa** is not referenced and may be **NULL**.

18: **iccola**$[dim]$ – Integer *Output*

Note: the dimension, *dim*, of the array **iccola** must be at least **maxn** $+ 1$ when **maxn** > 0.

On exit: a set of pointers to the beginning of each column of A. More precisely, **iccola**$[i-1]$ contains the index in **a** of the start of the ith column, for $i = 1, 2, \ldots,$ **n**. Note that **iccola**$[0] = 1$ and **iccola**[**n**] = **nnz** $+ 1$.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **iccola** is not referenced and may be **NULL**.

19: **bl**$[dim]$ – double *Output*

20: **bu**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the arrays **bl** and **bu** must be at least **maxn** + **maxm** when **maxn** > 0 and **maxm** > 0.

On exit: **bl** contains the vector l (the lower bounds) and **bu** contains the vector u (the upper bounds), for all the variables and constraints in the following order. The first **n** elements of each array contains the bounds on the variables x and the next **m** elements contains the bounds for the linear objective term $c^{\mathrm{T}}x$ and for the general linear constraints Ax (if any). Note that an 'infinite' lower bound is indicated by **bl**$[j-1] = -1.0\mathrm{e}+20$ and an 'infinite' upper bound by **bu**$[j-1] = +1.0\mathrm{e}+20$. In other words, any element of u greater than or equal to 10^{20} will be regarded as $+\infty$ (and similarly any element of l less than or equal to -10^{20} will be regarded as $-\infty$). If this value is deemed to be 'inappropriate', before calling nag_opt_sparse_convex_qp_solve (e04nqc) you are recommended to reset the value of its optional argument **Infinite Bound Size** and make any necessary changes to **bl** and/or **bu**.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **bl** and **bu** are not referenced and may be **NULL**.

21: **pnames**[**5**][9] – char *Input/Output*

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

pnames[0]
Must either contain the name of the problem or be blank.

pnames[1]
Must either be blank or contain the name of the objective row (in which case it overrides the OBJNAME section and the default choice of the first objective free row).

pnames[2]
Must either contain the name of the RHS set to be used or be blank (in which case the first RHS set is used).

pnames[3]
Must either contain the name of the RANGE set to be used or be blank (in which case the first RANGE set (if any) is used).

pnames[4]
Must either contain the name of the BOUNDS set to be used or be blank (in which case the first BOUNDS set (if any) is used).

On exit: a set of names associated with the problem as defined in the MPSX data file as follows:

pnames[0]
Contains the name of the problem (or blank if none).

pnames[1]
Contains the name of the objective row (or blank if none).

pnames[2]
Contains the name of the RHS set (or blank if none).

pnames[3]
Contains the name of the RANGE set (or blank if none).

pnames[4]
Contains the name of the BOUNDS set (or blank if none).

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **pnames** is not referenced and may be **NULL**.

22: **nname** – Integer * *Output*

On exit: $n + m$, the total number of variables and constraints in the problem (including the objective row).

If nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail.code** = NE_INT_MAX, **nname** is not set. In the former case you may pass **NULL** instead.

23: **crname**[*dim*][9] – char *Output*

Note: the dimension, *dim*, of the array **crname** must be at least **maxn** + **maxm** when **maxn** > 0 and **maxm** > 0.

On exit: the MPS names of all the variables and constraints in the problem in the following order. The first **n** elements contain the MPS names for the variables and the next **m** elements contain the MPS names for the objective row and general linear constraints (if any). Note that the MPS name for the objective row is stored in **crname**[**n** + **iobj** − 1].

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **crname** is not referenced and may be **NULL**.

24: **h**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **h** must be at least **maxnnzh** when **maxnnzh** > 0.

On exit: the **nnzh** nonzero elements of H, arranged by increasing column index.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **h** is not referenced and may be **NULL**.

25: **irowh**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **irowh** must be at least **maxnnzh** when **maxnnzh** > 0.

On exit: the **nnzh** row indices of the elements stored in H.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **irowh** is not referenced and may be **NULL**.

26: **iccolh**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **iccolh** must be at least **maxncolh** $+ 1$ when **maxncolh** > 0.

On exit: a set of pointers to the beginning of each column of H. More precisely, **iccolh**$[i-1]$ contains the index in H of the start of the ith column, for $i = 1, 2, \ldots,$ **ncolh**. Note that **iccolh**$[0] = 1$ and **iccolh**[**ncolh**] = **nnzh** $+ 1$.

If nag_opt_miqp_mps_read (e04mxc) is run in query mode (see Section 3.2), **iccolh** is not referenced and may be **NULL**.

27: **minmax** – Integer * *Output*

On exit: **minmax** defines the direction of the optimization as read from the MPS file. By default the function assumes the objective function should be minimized and will return **minmax** $= -1$. If the function discovers in the OBJSENSE section that the objective function should be maximized it will return **minmax** $= 1$. If the function discovers that there is neither the linear objective term c (the objective row) nor the Hessian matrix H, the problem is considered as a feasible point problem and **minmax** $= 0$ is returned.

If nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or returned with **fail**.**code** = NE_INT_MAX, **minmax** is not set. In the former case you may pass **NULL** instead.

28: **intvar**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **intvar** must be at least **maxlintvar**, when **maxlintvar** > 0.

On exit: if **maxlintvar** > 0 on entry, **intvar** contains pointers to the columns that are defined as integer variables. More precisely, **intvar**$[i-1] = k$, where k is the index of a column that is defined as an integer variable, for $i = 1, 2, \ldots,$ **lintvar**.

If **maxlintvar** ≤ 0 on entry, or nag_opt_miqp_mps_read (e04mxc) was run in query mode (see Section 3.2), or it returned with **fail**.**code** = NE_INT_MAX, **intvar** is not set. Excepting the latter case you may pass **NULL** as this argument instead.

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

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

Note that if any of the relevant arguments are accidentally set to zero, or not set and assume zero values, then the function will have executed in query mode. In this case only the size of the problem is returned and other arguments are not set. See Section 3.2.

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_FILEID

On entry, **fileid** = ⟨*value*⟩.
Constraint: **fileid** ≥ 0.

NE_INT_MAX

At least one of **maxm**, **maxn**, **maxnnz**, **maxnnzh**, **maxncolh** or **maxlintvar** is too small.
Suggested values are returned in **m**, **n**, **nnz**, **nnzh**, **ncolh** and **lintvar** respectively.

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 the Essential Introduction for further information.

NE_MPS_BOUNDS

The supplied name, in **pnames**[4], of the BOUNDS set to be used was not found in the BOUNDS section.

Unknown bound type '⟨*value*⟩' in BOUNDS section.

NE_MPS_COLUMNS

Column '⟨*value*⟩' has been defined more than once in the COLUMNS section. Column definitions must be continuous. (See Section 3.1.5).

Unknown column name '⟨*value*⟩' in ⟨*value*⟩ section.
All column names must be specified in the COLUMNS section.

NE_MPS_ENDATA_NOT_FOUND

End of file found before ENDATA indicator line.

NE_MPS_FORMAT

Warning: MPS file not strictly fixed format, although the problem was read anyway. The data may have been read incorrectly. You should set **mpslst** = 1 and repeat the call to nag_opt_miqp_mps_read (e04mxc) for more details.

NE_MPS_ILLEGAL_DATA_LINE

An illegal line was detected in '⟨*value*⟩' section.
This is neither a comment nor a valid data line.

NE_MPS_ILLEGAL_NUMBER

Field ⟨*value*⟩ did not contain a number (see Section 3).

NE_MPS_INDICATOR

Incorrect ordering of indicator lines.
BOUNDS indicator line found before COLUMNS indicator line.

Incorrect ordering of indicator lines.
COLUMNS indicator line found before ROWS indicator line.

Incorrect ordering of indicator lines.
OBJNAME indicator line found after ROWS indicator line.

Incorrect ordering of indicator lines.
QUADOBJ indicator line found before BOUNDS indicator line.

Incorrect ordering of indicator lines.
QUADOBJ indicator line found before COLUMNS indicator line.

Incorrect ordering of indicator lines.
RANGES indicator line found before RHS indicator line.

Incorrect ordering of indicator lines.
RHS indicator line found before COLUMNS indicator line.

Indicator line '⟨*value*⟩' has been found more than once in the MPS file.

No indicator line found in file. It may be an empty file.

Unknown indicator line '⟨*value*⟩'.

NE_MPS_INVALID_INTORG_INTEND

Found '`INTEND`' marker without previous marker being '`INTORG`'.

Found '`INTORG`' but not '`INTEND`' before the end of the COLUMNS section.

Found '`INTORG`' marker within '`INTORG`' to '`INTEND`' range.

Illegal marker type '⟨*value*⟩'.
Should be either '`INTORG`' or '`INTEND`'.

NE_MPS_MANDATORY

At least one mandatory section not found in MPS file.

NE_MPS_OBJNAME

The supplied name, in **pnames**[1] or in OBJNAME, of the objective row was not found among the free rows in the ROWS section.

NE_MPS_PRINTABLE

Illegal column name.
Column names must consist of printable characters only.

Illegal row name.
Row names must consist of printable characters only.

NE_MPS_RANGES

The supplied name, in **pnames**[3], of the RANGES set to be used was not found in the RANGES section.

NE_MPS_REPEAT_COLUMN

More than one nonzero of **a** has row name '⟨*value*⟩' and column name '⟨*value*⟩' in the COLUMNS section.

NE_MPS_REPEAT_ROW

Row name '⟨*value*⟩' has been defined more than once in the ROWS section.

NE_MPS_RHS

The supplied name, in **pnames**[2], of the RHS set to be used was not found in the RHS section.

NE_MPS_ROWS

Unknown inequality key '⟨*value*⟩' in ROWS section.
Expected '`N`', '`G`', '`L`' or '`E`'.

Unknown row name '⟨*value*⟩' in ⟨*value*⟩ section.
All row names must be specified in the ROWS section.

NE_MPS_ROWS_OR_CONS

Empty ROWS section.
Neither the objective row nor the constraints were defined.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_miqp_mps_read (e04mxc) is not threaded by NAG in any implementation.

nag_opt_miqp_mps_read (e04mxc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example solves the quadratic programming problem

$$\text{minimize} \quad c^{\mathrm{T}}x + \tfrac{1}{2}x^{\mathrm{T}}Hx \quad \text{subject to} \quad \begin{array}{rcl} l & \leq Ax & \leq u, \\ -2 & \leq x & \leq 2, \end{array}$$

where

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

$$A = \begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & -2.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 1.0 & -1.0 & 1.0 & -1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{pmatrix},$$

$$l = \begin{pmatrix} -2.0 \\ -2.0 \\ -2.0 \end{pmatrix} \quad \text{and} \quad u = \begin{pmatrix} 1.5 \\ 1.5 \\ 4.0 \end{pmatrix}.$$

The optimal solution (to five figures) is

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

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

The MPS representation of the problem is given in Section 10.2.

10.1 Program Text

```
/* nag_opt_miqp_mps_read (e04mxc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#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 e04mx_name[9];

int main(void)
{
  Integer exit_status = 0;
  double obj, objadd, sinf;
  Integer i, iobj, j, lenc, lintvar, m, maxlintvar, maxm, maxn, maxncolh,
    maxnnz, maxnnzh, minmax, n, ncolh, ninf, nname, nnz, nnzh, ns;
  Integer mpslst = 1;
  double *a = 0, *bl = 0, *bu = 0, *h = 0, *pi = 0, *rc = 0, *ruser = 0, *x = 0;
```

```
  Integer *helast = 0, *hs = 0, *iccola = 0, *iccolh = 0, *intvar = 0,
    *irowa = 0, *irowh = 0, *iuser = 0;
  char pnames[5][9] = {"", "", "", "", ""};
  char (*crname)[9] = 0;
  char **names = 0;
  char fname[] = "e04mxce.opt";

  /* Nag Types */
  Nag_Boolean readints = Nag_FALSE;
  Nag_E04State state;
  NagError fail;
  Nag_Comm comm;
  Nag_FileID fileid;

  INIT_FAIL(fail);

  printf("nag_opt_miqp_mps_read (e04mxc) Example Program Results\n");
  fflush(stdout);

  /* nag_open_file (x04acc).
     Open unit number for reading and associate unit with named file. */
  nag_open_file(fname, 0, &fileid, NAGERR_DEFAULT);

  /* nag_opt_miqp_mps_read (e04mxc).
     Reads MPS data file defining LP, QP, MILP or MIQP problem.
     Query call. */
  nag_opt_miqp_mps_read(fileid, 0, 0, 0, 0, 0, 0, mpslst, &n, &m, &nnz, &ncolh,
                        &nnzh, &lintvar, NULL, NULL, NULL, NULL, NULL, NULL,
                        NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                        NAGERR_DEFAULT);

  /* nag_close_file (x04adc).
     Close file associated with given unit number. */
  nag_close_file(fileid, NAGERR_DEFAULT);

  maxm = m;
  maxn = n;
  maxnnz = nnz;
  maxnnzh = nnzh;
  maxncolh = ncolh;
  maxlintvar = (readints && lintvar > 0) ? lintvar : -1;

  if (!(irowa = NAG_ALLOC(maxnnz, Integer)) ||
      !(iccola = NAG_ALLOC(maxn + 1, Integer)) ||
      !(a = NAG_ALLOC(maxnnz, double)) ||
      !(bl = NAG_ALLOC(maxn + maxm, double)) ||
      !(bu = NAG_ALLOC(maxn + maxm, double)) ||
      !(irowh = NAG_ALLOC(maxnnzh, Integer)) ||
      !(iccolh = NAG_ALLOC(maxncolh + 1, Integer)) ||
      !(h = NAG_ALLOC(maxnnzh, double)) ||
      (maxlintvar > 0 && !(intvar = NAG_ALLOC(maxlintvar, Integer))) ||
      !(crname = NAG_ALLOC(maxn + maxm, e04mx_name)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  nag_open_file(fname, 0, &fileid, NAGERR_DEFAULT);

  /* Full call to the reader. */
  nag_opt_miqp_mps_read(fileid, maxn, maxm, maxnnz, maxncolh, maxnnzh,
                        maxlintvar, mpslst, &n, &m, &nnz, &ncolh, &nnzh,
                        &lintvar, &iobj, a, irowa, iccola, bl, bu, pnames,
                        &nname, crname, h, irowh, iccolh, &minmax,
                        (maxlintvar > 0) ? intvar : NULL, NAGERR_DEFAULT);

  nag_close_file(fileid, NAGERR_DEFAULT);

  /* Data has been read. Set up and run the solver.
     We have no explicit objective vector so set lenc = 0; the
```

```
   objective vector is stored in row iobj of a. */
lenc = 0;
objadd = 0.0;

if (!(helast = NAG_ALLOC(n + m, Integer)) ||
    !(x = NAG_ALLOC(n + m, double)) ||
    !(pi = NAG_ALLOC(m, double)) ||
    !(rc = NAG_ALLOC(n + m, double)) ||
    !(hs = NAG_ALLOC(n + m, Integer)) ||
    !(iuser = NAG_ALLOC(ncolh + 1 + nnzh, Integer)) ||
    !(ruser = NAG_ALLOC(nnzh, double)) ||
    !(names = NAG_ALLOC(n + m, char *)))
  {
    printf("Allocation failure\n");
    exit_status = -3;
    goto END;
  }

/* Transform char (*crname)[9] to char *names[] to be compatible with the
   solver. */
for (i = 0; i < n + m; i++)
  names[i] = crname[i];

for (i = 0; i < n + m; i++)
  helast[i] = 0;
for (i = 0; i < n + m; i++)
  hs[i] = 0;
for (i = 0; i < n + m; i++)
  x[i] = MIN(MAX(0.0, bl[i]), bu[i]);

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);

/* Use nag_opt_sparse_convex_qp_option_set_string (e04nsc) to change
   the direction of optimization. Minimization is assumed by default. */
if (minmax == 1)
  nag_opt_sparse_convex_qp_option_set_string("Maximize", &state,
                                             NAGERR_DEFAULT);
else if (minmax == 0)
  nag_opt_sparse_convex_qp_option_set_string("Feasible Point", &state,
                                             NAGERR_DEFAULT);

/* 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 and then call
   nag_opt_sparse_convex_qp_option_set_integer (e04ntc) to register that
   setting with the solver. */
nag_open_file("", 2, &fileid, NAGERR_DEFAULT);
nag_opt_sparse_convex_qp_option_set_integer("Print file", fileid, &state,
                                            NAGERR_DEFAULT);

/* nag_opt_sparse_convex_qp_solve (e04nqc).
   LP or QP problem (suitable for sparse problems). */
nag_opt_sparse_convex_qp_solve(Nag_Cold, qphx, m, n, nnz, nname, lenc,
                               ncolh, iobj, objadd, pnames[0], a, irowa,
                               iccola, bl, bu, NULL, (const char**)names,
```

```
                                helast, hs, x, pi, rc, &ns, &ninf, &sinf,
                                &obj, &state, &comm, &fail);

 END:
  NAG_FREE(a);
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(h);
  NAG_FREE(pi);
  NAG_FREE(rc);
  NAG_FREE(ruser);
  NAG_FREE(x);
  NAG_FREE(helast);
  NAG_FREE(hs);
  NAG_FREE(iccola);
  NAG_FREE(iccolh);
  NAG_FREE(intvar);
  NAG_FREE(irowa);
  NAG_FREE(irowh);
  NAG_FREE(iuser);
  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

```
NAME          E04MX.EX
ROWS
 L  ..ROW1..
 L  ..ROW2..
 L  ..ROW3..
 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
    RHS1      ..COST..      1000.0
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
QUADOBJ
    ...X1...  ...X1...  2.00000000E0   ...X2...  1.00000000E0
    ...X1...  ...X3...  1.00000000E0   ...X4...  1.00000000E0
    ...X1...  ...X5...  1.00000000E0
    ...X2...  ...X2...  2.00000000E0   ...X3...  1.00000000E0
    ...X2...  ...X4...  1.00000000E0   ...X5...  1.00000000E0
    ...X3...  ...X3...  2.00000000E0   ...X4...  1.00000000E0
    ...X3...  ...X5...  1.00000000E0
    ...X4...  ...X4...  2.00000000E0   ...X5...  1.00000000E0
    ...X5...  ...X5...  2.00000000E0
ENDATA
```

10.3 Program Results

```
nag_opt_miqp_mps_read (e04mxc) Example Program Results

 MPSX INPUT LISTING
 ------------------
 Searching for indicator line
 Line          1: Found NAME indicator line
                  Query mode - Ignoring NAME data.
 Line          2: Found ROWS indicator line
                  Query mode - Counting ROWS data.
 Line          7: Found COLUMNS indicator line
                  Query mode - Counting COLUMNS data.
 Line         26: Found RHS indicator line
                  Query mode - Ignoring RHS data.
 Line         31: Found RANGES indicator line
                  Query mode - Ignoring RANGES data.
 Line         35: Found BOUNDS indicator line
                  Query mode - Counting BOUNDS data.
 Line         54: Found QUADOBJ indicator line
                  Query mode - Counting QUADOBJ data.
```

```
                Query mode - End of QUADOBJ data. Exit

MPSX INPUT LISTING
------------------
Searching for indicator line
Line          1: Found NAME indicator line
Line          2: Found ROWS indicator line
Line          7: Found COLUMNS indicator line
Line         26: Found RHS indicator line
Line         31: Found RANGES indicator line
Line         35: Found BOUNDS indicator line
Line         54: Found QUADOBJ indicator line
Line         64: Found ENDATA indicator line

Parameters
==========

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

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

LP/QP Parameters
----------------
Minimize...............                 QPsolver Cholesky......                 Cold start.............
Scale tolerance........     0.900       Feasibility tolerance..  1.00E-06       Iteration limit........     10000
Scale option...........         2       Optimality tolerance...  1.00E-06       Print level............         1
Crash tolerance........     0.100       Pivot tolerance........  2.04E-11       Partial price..........         1
Crash option...........         3       Elastic weight.........  1.00E+00       Prtl price section ( A)         9
Elastic mode...........         1       Elastic objective......         1       Prtl price section (-I)         4

QP objective
------------
Objective variables....         5       Hessian columns........         5       Superbasics limit......         6
Nonlin Objective vars..         5       Unbounded step size....  1.00E+20
Linear Objective vars..         0

Miscellaneous
-------------
LU factor tolerance....      3.99       LU singularity tol.....  2.04E-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               4           0           1           0           3
Columns            9           0           0           0           9

No. of matrix elements                   36     Density     100.000
Biggest                           4.0000E+00  (excluding fixed columns,
Smallest                          1.0000E+00   free rows, and RHS)

No. of objective coefficients             9
Biggest                           4.0000E+00  (excluding fixed columns)
Smallest                          1.0000E-01

Nonlinear constraints       0     Linear constraints       4
Nonlinear variables         5     Linear variables         4
Jacobian  variables         0     Objective variables      5
Total constraints           4     Total variables          9
```

```
Itn      0: Feasible linear constraints

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

Problem name                 E04MX.EX
No. of iterations                  11   Objective value     -8.0677777778E+00
No. of Hessian products            25   Objective row       -1.0785555556E+01
                                        Quadratic objective  2.7177777778E+00
No. of superbasics                  4   No. of basic nonlinears             2
No. of degenerate steps             2   Percentage                      18.18
Max x       (scaled)        1 1.3E+00   Max pi      (scaled)        4 1.0E+00
Max x                       1 2.0E+00   Max pi                      4 1.0E+00
Max Prim inf(scaled)        0 0.0E+00   Max Dual inf(scaled)        0 0.0E+00
Max Primal infeas           0 0.0E+00   Max Dual infeas             0 0.0E+00

Name           E04MX.EX                 Objective Value     -8.0677777778E+00

Status         Optimal Soln             Iteration     11    Superbasics     4

Section 1 - Rows

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

     10  ..ROW1..    UL         1.50000          .             -2.00000         1.50000        -0.06667      1
     11  ..ROW2..    UL         1.50000          .             -2.00000         1.50000        -0.03333      2
     12  ..ROW3..   SBS         3.93333        -0.06667        -2.00000         4.00000          .           3
     13  ..COST..    BS       -10.78556       -10.78556           None           None         -1.0          4

Section 2 - Columns

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

      1  ...X1...    UL         2.00000        -0.90000        -2.00000         2.00000        -0.80000      5
      2  ...X2...   SBS        -0.23333        -0.13333        -2.00000         2.00000          .           6
      3  ...X3...    BS        -0.26667        -0.16667        -2.00000         2.00000          .           7
      4  ...X4...    BS        -0.30000        -0.20000        -2.00000         2.00000          .           8
      5  ...X5...   SBS        -0.10000          .             -2.00000         2.00000          .           9
      6  ...X6...    UL         2.00000        -1.0            -2.00000         2.00000        -0.90000     10
      7  ...X7...    UL         2.00000        -1.0            -2.00000         2.00000        -0.90000     11
      8  ...X8...   SBS        -1.77778        -0.10000        -2.00000         2.00000          .          12
      9  ...X9...    BS        -0.45556        -0.30000        -2.00000         2.00000          .          13
```