NAG Library Function Document

nag_ip_mps_read (h02buc)

1 Purpose

nag_ip_mps_read (h02buc) reads data for a linear, quadratic, or integer programming problem from a file which is in standard or compatible MPSX input format.

2 Specification

```
#include <nag.h>
#include <nagh.h>
void nag_ip_mps_read (const char *mps_file, Nag_Boolean minimize, Integer *n,
     Integer *m, double **a, double **bl, double **bu, Nag_Boolean **intvar,
     double **cvec, double **x, Nag_H02_Opt *options, NagError *fail)
```

3 Description

nag_ip_mps_read (h02buc) reads linear programming (LP), linear terms of quadratic programming (QP), or integer programming (IP) problem data from a file which is prepared in standard or compatible MPSX (IBM (1971)) 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 for use with functions which are designed to solve problems of the form

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

where $f(x)$ is a linear function (of the form $c^{\mathrm{T}}x$) or a quadratic function (of the form $c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$). Note that for quadratic problems, nag_ip_mps_read (h02buc) reads only the linear part of the objective; the quadratic part must be supplied separately to the solver. See the documentation for the appropriate solver for further details. (nag_ip_mps_read (h02buc) is primarily designed for use with nag_ip_bb (h02bbc), but may also be used in conjunction with nag_opt_lp (e04mfc), nag_opt_lin_lsq (e04ncc) and nag_opt_qp (e04nfc)).

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_ip_mps_read (h02buc) are all allocated internally.

MPSX Input Format

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

1. Indicator lines (specifying the type of data which is to follow).
2. 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_ip_mps_read (h02buc) 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 function. 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., $=$, $\geq$ or $\leq$).

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., $\geq$).

`L` less than or equal to (i.e., $\leq$).

`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 $c^{\mathrm{T}}x$. It is taken to be the first free row, unless some other free row name is specified by the optional argument **options.obj_name** (see Section 11.2). 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 nonzero elements of A and c need to be specified in the COLUMNS section, as any unspecified elements of A and c are assumed to be zero. In addition, any nonzero elements in the jth column of A must be grouped together before those in the $(j+1)$th column, for $j = 1, 2, \ldots, \mathbf{n} - 1$. Nonzero 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 nonzero 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 $l \leq Ax \leq u$, 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 summarised 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 $l \leq x \leq u$). 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 arguments **options.col_lo_default** and **options.col_up_default**; see Section 11.2.) 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.

Integer Programming Problems

In IP problems there are two common integer variable types: (a) 0–1 integer variables (or 'binary' variables) which represent 'on' or 'off' situations and (b) general integer variables which are forced to take an integer value, in a specified range, at the optimal integer solution. Integer variables can be defined in the following compatible and standard MPSX forms.

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_ip_mps_read (h02buc) accepts both standard and/or compatible MPSX format as a means of specifying integer variables.

An example of the compatible MPSX format is given in Section 9 and an example of the standard MPSX format is given in Section 12.

4 References

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

5 Arguments

1: **mps_file** – const char * *Input*

On entry: 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`.

2: **minimize** – Nag_Boolean *Input*

On entry: specifies the direction of the optimization. **minimize** must be set to Nag_TRUE for minimization and to Nag_FALSE for maximization. For a maximization problem, c, the coefficients of the linear part of the objective function, is negated with respect to its definition in the MPSX file. For maximization problems involving a quadratic objective function, you must also modify the sign of the quadratic term as appropriate.

3: **n** – Integer * *Output*

On exit: n, the number of variables specified by the data file.

4: **m** – Integer * *Output*

On exit: m, the number of general linear constraints specified by the data file.

5: **a** – double ** *Output*

On exit: A, the matrix of general linear constraints.

Sufficient memory is allocated internally by nag_ip_mps_read (h02buc) and may be freed by the utility function nag_ip_mps_free (h02bvc).

6: **bl** – double ** *Output*

7: **bu** – double ** *Output*

On exit: **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 general linear constraints Ax (if any). Note that an 'infinite' lower bound is indicated by $\mathbf{bl}[j-1] = -10^{20}$, an 'infinite' upper bound by $\mathbf{bu}[j-1] = 10^{20}$, and an equality constraint by $\mathbf{bl}[j-1] = \mathbf{bu}[j-1]$.

Sufficient memory is allocated internally by nag_ip_mps_read (h02buc) and may be freed by the utility function nag_ip_mps_free (h02bvc).

8: **intvar** – Nag_Boolean ** *Output*

On exit: indicates which are the integer variables in the problem. More precisely, $\mathbf{intvar}[j-1] =$ Nag_TRUE if x_j is an integer variable, and Nag_FALSE otherwise, for $j = 1, 2, \ldots, n$.

Sufficient memory is allocated internally by nag_ip_mps_read (h02buc) and may be freed by the utility function nag_ip_mps_free (h02bvc).

9: **cvec** – double ** *Output*

On exit: c, the coefficients of the linear term of the objective function. The signs of these coefficients are determined by the problem and the direction of the optimization (see **minimize** above).

Sufficient memory is allocated internally by nag_ip_mps_read (h02buc) and may be freed by the utility function nag_ip_mps_free (h02bvc).

10: **x** – double ** *Output*

On exit: an initial estimate of the solution to the problem. More precisely, $\mathbf{x}[j] = \min(\max(0.0, \mathbf{bl}[j]), \mathbf{bu}[j])$, for $j = 0, 1, \ldots, n-1$.

Sufficient memory is allocated internally by nag_ip_mps_read (h02buc) and may be freed by the utility function nag_ip_mps_free (h02bvc).

11: **options** – Nag_H02_Opt * *Input/Output*

On entry/exit: a pointer to a structure of type Nag_H02_Opt whose members are optional arguments for nag_ip_mps_read (h02buc). These structure members offer the means of adjusting the argument 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 11.2.

If any of these optional arguments are required then the structure **options** should be declared and initialized by a call to nag_ip_init (h02xxc) and supplied as an argument to nag_ip_mps_read (h02buc). However, if the optional arguments are not required the NAG defined null pointer, `H02_DEFAULT`, can be used in the function call.

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

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

5.1 Description of Printed Output

Results are printed out by default. The level of printed output can be controlled with the structure members **options**.**list** and **options**.**output_level** (see Section 11.2). If **options**.**list** = Nag_TRUE then the argument values to nag_ip_mps_read (h02buc) are listed, whereas the printout of results is governed by the value of **options**.**output_level**. The default, **options**.**output_level** = 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.

(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 arguments **options**.**prob_name**, **options**.**obj_name**, **options**.**rhs_name**, **options**.**range_name** and/or **options**.**bnd_name** (see Section 11), 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 **options**.**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 **options**.**output_level** = Nag_NoOutput then printout will be suppressed; you can print the information contained in (b) and (c) when nag_ip_mps_read (h02buc) returns to the calling program.

6 Error Indicators and Warnings

NE_2_REAL_EE_OPT_ARG_CONS

On entry, **options**.**col_lo_default** = ⟨*value*⟩ while **options**.**col_up_default** = ⟨*value*⟩. Constraint: **options**.**col_lo_default** ≤ **options**.**col_up_default**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **options**.**bnd_name** had an illegal value.

On entry, argument **options**.**obj_name** had an illegal value.

On entry, argument **options**.**output_level** had an illegal value.

On entry, argument **options**.**prob_name** had an illegal value.

On entry, argument **options**.**range_name** had an illegal value.

On entry, argument **options**.**rhs_name** had an illegal value.

NE_INT_OPT_ARG_LT

On entry, **options**.**ncol_approx** = ⟨*value*⟩. Constraint: **options**.**ncol_approx** ≥ 1.

On entry, **options**.**nrow_approx** = ⟨*value*⟩. Constraint: **options**.**nrow_approx** ≥ 1.

NE_INTERNAL_ERROR

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

NE_MPS_ENDATA_NOT_FOUND

The file does not contain an ENDATA indicator.

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 ⟨*value*⟩: ⟨*string*⟩.

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 ⟨*value*⟩: ⟨*string*⟩.

NE_MPS_ILLEGAL_NUMBER

Number expected but value could not be read. Check numerical fields. Error at MPSX line ⟨*value*⟩: ⟨*string*⟩.

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 ⟨*value*⟩: ⟨*string*⟩.

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_PROB_NOT_FOUND

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

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_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_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 **options**.**crnames**.

NE_NOT_APPEND_FILE

Cannot open file ⟨*string*⟩ for appending.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*string*⟩.

NE_NOT_READ_FILE

Cannot open file ⟨*string*⟩ for reading.

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 **a** is a null pointer. It should contain the address of a variable of type double *.
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 **intvar** is a null pointer. It should contain the address of a variable of type Boolean *.
Argument **cvec** is a null pointer. It should contain the address of a variable of type double *.
Argument **x** is a null pointer. It should contain the address of a variable of type double *.

NE_OPT_NOT_INIT

Options structure not initialized.

NE_WRITE_ERROR

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Although nag_ip_mps_read (h02buc) is designed primarily for use with nag_ip_bb (h02bbc), it can also be used in conjunction with nag_opt_lp (e04mfc) (as illustrated by Section 10), nag_opt_lin_lsq (e04ncc) and nag_opt_qp (e04nfc). However, these last three functions do not provide a direct means of using the row and column names which can be read by nag_ip_mps_read (h02buc) and stored in the optional argument **options**.**crnames**. By making use of the user-defined printing facilities of the functions, you can customize the solution printing to print the row and column names (see Section 11.2 in nag_ip_bb (h02bbc)). Alternatively, you may call nag_ip_bb (h02bbc) to solve the LP or QP problem by specifying all variables to be non-integer via the **intvar** argument (see Section 5 in nag_ip_bb (h02bbc)).

10 Example

This example reads in a compatible MPSX file (see Section 3 for a description of standard and compatible MPSX formats) which specifies an instance of the so-called diet problem, and solves it as an IP problem.

The example calls nag_ip_init (h02xxc), which initializes the **options** structure and nag_ip_read (h02xyc) which reads optional argument settings from the data file. The argument settings suppress all output from nag_ip_mps_read (h02buc). The program then calls nag_ip_mps_read (h02buc) to read the MPSX data. The program then sets the optional argument **options**.**list** to Nag_TRUE before calling nag_ip_bb (h02bbc) to solve the IP problem. As the **options** structure is passed as an argument, the row and column names read from the file are used in the solution output (see Section 10.3).

Finally, nag_ip_mps_free (h02bvc) is called to free the problem arrays, and nag_ip_free (h02xzc) is called to free the memory in **options**.

10.1 Program Text

```
/* nag_ip_mps_read (h02buc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 * Mark 6 revised, 2000.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <string.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagh02.h>

int main(void)
{
  const char  *optionsfile = "h02buce.opt";
  Integer     exit_status = 0;
  Nag_Boolean *intvar;
  Integer     m, n;
  Nag_H02_Opt options;
  double      *a, *bl, *bu, *c, objf, *x;
  Nag_Comm    comm;
  NagError    fail;
```

```
  INIT_FAIL(fail);


  printf("nag_ip_mps_read (h02buc) Example Program Results\n");

  /* Initialise options structure and read option settings. */
  /* nag_ip_init (h02xxc).
   * Initialize option structure to null values
   */
  nag_ip_init(&options);
  /* nag_ip_read (h02xyc).
   * Read optional parameter values from a file
   */
  fflush(stdout);
  nag_ip_read("h02buc", optionsfile, &options, (Nag_Boolean) Nag_TRUE,
              "stdout", &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ip_read (h02xyc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Read MPSX data */
  /* nag_ip_mps_read (h02buc), see above. */
  nag_ip_mps_read( 0, Nag_TRUE, &n, &m, &a, &bl, &bu, &intvar, &c, &x,
                  &options, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ip_mps_read (h02buc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Solve IP problem defined by data */
  options.list = Nag_TRUE;
  /* nag_ip_bb (h02bbc).
   * Solves integer programming problems using a branch and
   * bound method
   */
  nag_ip_bb(n, m, a, n, bl, bu, intvar, c, (double *) 0, 0, NULLFN,
            x, &objf, &options, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ip_bb (h02bbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Free memory allocated by nag_ip_mps_read (h02buc) */
  /* nag_ip_mps_free (h02bvc), see above. */
  nag_ip_mps_free(&a, &bl, &bu, &intvar, &c, &x);

  /* Free options memory */
  /* nag_ip_free (h02xzc).
   * Free NAG allocated memory from option structures
   */
  nag_ip_free(&options, "all", &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ip_free (h02xzc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

 END:

  return exit_status;
}
```

10.2 Program Data

```
* nag_ip_mps_read (h02buc) Example Program Data
*
* This is an example of 'standard format' MPSX data.
*
NAME          DIET2
ROWS
 G  ENERGY
 G  PROTEIN
 G  CALCIUM
 N  COST
COLUMNS
    OATMEAL   ENERGY      110.0
    OATMEAL   PROTEIN     4.0
    OATMEAL   CALCIUM     2.0
    OATMEAL   COST        3.0
    CHICKEN   ENERGY      205.0
    CHICKEN   PROTEIN     32.0
    CHICKEN   CALCIUM     12.0
    CHICKEN   COST        24.0
    INTEGER   'MARKER'                'INTORG'
    EGGS      ENERGY      160.0
    EGGS      PROTEIN     13.0
    EGGS      CALCIUM     54.0
    EGGS      COST        13.0
    MILK      ENERGY      160.0
    MILK      PROTEIN     8.0
    MILK      CALCIUM     285.0
    MILK      COST        9.0
    PIE       ENERGY      420.0
    PIE       PROTEIN     4.0
    PIE       CALCIUM     22.0
    PIE       COST        20.0
    INTEGER   'MARKER'                'INTEND'
    BACON     ENERGY      260.0
    BACON     PROTEIN     14.0
    BACON     CALCIUM     80.0
    BACON     COST        19.0
RHS
    DEMANDS   ENERGY      2000.0
    DEMANDS   PROTEIN     55.0
    DEMANDS   CALCIUM     800.0
BOUNDS
 UI SERVINGS  OATMEAL     4.0
 UI SERVINGS  CHICKEN     3.0
 UP SERVINGS  EGGS        2.0
 UP SERVINGS  MILK        8.0
 UP SERVINGS  PIE         2.0
 UI SERVINGS  BACON       2.0
ENDATA
```

```
nag_ip_mps_read (h02buc) Example Program Optional Parameters
  BEGIN nag_ip_mps_read

    list = Nag_FALSE
    output_level = Nag_NoOutput
    prob_name = DIET2

  END
```

10.3 Program Results

```
nag_ip_mps_read (h02buc) Example Program Results

Optional parameter setting for h02buc.
--------------------------------------

Option file: h02buce.opt

list set to Nag_FALSE
```

```
output_level set to Nag_NoOutput
prob_name set to DIET2

Parameters to h02bbc
--------------------

Linear constraints.............  3    Number of variables...........  6
Number of integer variables...  6

prob..................... Nag_MILP
feas_tol................. 1.05e-08    machine precision....... 1.11e-16
inf_bound................ 1.00e+20    max_iter.................       50
first_soln.............. Nag_FALSE    max_depth................       10
max_nodes.............ALL_NODES    int_tol.................. 1.00e-05
int_obj_bound........... 1.00e+20    soln_tol................. 1.05e-08
nodsel..........Nag_MinObj_Search    varsel..............Nag_First_Int
branch_dir........Nag_Branch_Left    crnames..................supplied
print_level.........Nag_Soln_Iter
outfile.................   stdout

Memory allocation:
lower....................      Nag
upper....................      Nag
state....................      Nag
lambda...................      Nag

  Node Parent     Obj     Varbl     Value     Lower     Upper      Value Depth
   No   Node     Value   Chosen    Before     Bound     Bound      After
    1         9.250e+01
    2      1  9.385e+01        4  4.50e+00  0.00e+00  4.00e+00   4.00e+00     1
    3      1  9.319e+01        4  4.50e+00  5.00e+00  8.00e+00   5.00e+00     1
    4      3  9.612e+01        5  1.81e+00  0.00e+00  1.00e+00   1.00e+00     2
    5      3  9.482e+01        5  1.81e+00  2.00e+00  2.00e+00   2.00e+00     2
    6      2  9.450e+01        6  3.08e-01  0.00e+00  0.00e+00   0.00e+00     2
    7      2  9.688e+01        6  3.08e-01  1.00e+00  2.00e+00   1.00e+00     2
    8      6  9.737e+01        3  5.00e-01  0.00e+00  0.00e+00   0.00e+00     3
    9      6  9.650e+01        3  5.00e-01  1.00e+00  2.00e+00   1.00e+00     3
   10      5  9.569e+01        1  3.27e+00  0.00e+00  3.00e+00   3.00e+00     3
   11      5  9.700e+01        1  3.27e+00  4.00e+00  4.00e+00   4.00e+00     3
   *** Integer Solution ***

   12     10  9.619e+01        4  5.19e+00  5.00e+00  5.00e+00   5.00e+00     4
   13     10  9.945e+01 CO     4  5.19e+00  6.00e+00  8.00e+00   6.00e+00     4
   14      4  9.646e+01        4  7.12e+00  5.00e+00  7.00e+00   7.00e+00     3
   15      4  9.733e+01 CO     4  7.12e+00  8.00e+00  8.00e+00   8.00e+00     3
   16     12  9.644e+01        6  1.15e-01  0.00e+00  0.00e+00   0.00e+00     5
   17     12  1.067e+02 CO     6  1.15e-01  1.00e+00  2.00e+00   1.00e+00     5
   18     16  9.751e+01 CO     3  1.88e-01  0.00e+00  0.00e+00   0.00e+00     6
   19     16  1.035e+02 CO     3  1.88e-01  1.00e+00  2.00e+00   1.00e+00     6
   20     14  9.662e+01        6  7.69e-02  0.00e+00  0.00e+00   0.00e+00     4
   21     14  1.005e+02 CO     6  7.69e-02  1.00e+00  2.00e+00   1.00e+00     4
   22      9  9.850e+01 CO     4  3.50e+00  0.00e+00  3.00e+00   3.00e+00     4
   23      9  9.719e+01 CO     4  3.50e+00  4.00e+00  4.00e+00   4.00e+00     4
   24     20  9.734e+01 CO     3  1.25e-01  0.00e+00  0.00e+00   0.00e+00     5

  Node Parent     Obj     Varbl     Value     Lower     Upper      Value Depth
   No   Node     Value   Chosen    Before     Bound     Bound      After
   25     20  1.001e+02 CO     3  1.25e-01  1.00e+00  2.00e+00   1.00e+00     5
   26      7  1.052e+02 CO     4  2.88e+00  0.00e+00  2.00e+00   2.00e+00     3
   27      7  9.705e+01 CO     4  2.88e+00  3.00e+00  4.00e+00   3.00e+00     3

Final solution:

Varbl   State    Value       Lower Bound  Upper Bound    Lagr Mult    Residual

OATMEAL   EQ   4.00000e+00    4.0000e+00   4.0000e+00    3.000e+00   0.000e+00
CHICKEN   LL   0.00000e+00    0.0000e+00   3.0000e+00    2.400e+01   0.000e+00
EGGS      LL   0.00000e+00    0.0000e+00   2.0000e+00    1.300e+01   0.000e+00
MILK      LL   5.00000e+00    5.0000e+00   8.0000e+00    9.000e+00   0.000e+00
PIE       EQ   2.00000e+00    2.0000e+00   2.0000e+00    2.000e+01   0.000e+00
BACON     LL   0.00000e+00    0.0000e+00   2.0000e+00    1.900e+01   0.000e+00
```

```
Constr  State    Value       Lower Bound  Upper Bound    Lagr Mult    Residual

ENERGY    FR   2.08000e+03    2.0000e+03      None       0.000e+00   8.000e+01
PROTEIN   FR   6.40000e+01    5.5000e+01      None       0.000e+00   9.000e+00
CALCIUM   FR   1.47700e+03    8.0000e+02      None       0.000e+00   6.770e+02

Exit from branch and bound tree search after 27 nodes.

Optimal IP solution found.

Final IP objective value =   9.7000000e+01
```

11 Optional Arguments

A number of optional input and output arguments to nag_ip_mps_read (h02buc) are available through the structure argument **options**, type Nag_H02_Opt. An argument may be selected by assigning an appropriate value to the relevant structure member; those arguments not selected will be assigned default values. If no use is to be made of any of the optional arguments you should use the NAG defined null pointer, `H02_DEFAULT`, in place of **options** when calling nag_ip_mps_read (h02buc); the default settings will then be used for all arguments.

Before assigning values to **options** directly the structure **must** be initialized by a call to the function nag_ip_init (h02xxc). 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_ip_read (h02xyc) 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.

11.1 Optional Argument Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag_ip_mps_read (h02buc) together with their default values where relevant.

`Boolean list`	Nag_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`	100
`Integer nrow_approx`	100
`char **crnames`	size $\mathbf{n}+\mathbf{m}$
`Integer n_ivar`	

11.2 Description of the Optional Arguments

list – Nag_Boolean Default = Nag_TRUE

On entry: if **options**.**list** = Nag_TRUE the argument settings in the call to nag_ip_mps_read (h02buc) will be printed.

output_level – Nag_OutputType Default = Nag_MPS_Summary

On entry: the level of printout produced by nag_ip_mps_read (h02buc). 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 – const char[80] Default = `stdout`

On entry: 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 Default: **options.prob_name**[0] = '\0'
obj_name – char Default: **options.obj_name**[0] = '\0'
rhs_name – char Default: **options.rhs_name**[0] = '\0'
range_name – char Default: **options.range_name** = '\0'
bnd_name – char Default: **options.bnd_name**[0] = '\0'

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

options.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 **options.prob_name** is not blank, then nag_ip_mps_read (h02buc) will search the MPSX file for the specified problem. If **options.prob_name** is blank, then the first problem encountered will be read.

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

options.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).

options.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).

options.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).

Constraint: 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.

On exit: 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_ip_mps_read (h02buc) (see Section 6).

col_lo_default – double Default = 0.0

On entry: 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}

On entry: 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 $= 100$
nrow_approx – Integer Default $= 100$

On entry: an estimate of the number of columns and rows in the problem. nag_ip_mps_read (h02buc) 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.

Constraints:

options.**ncol_approx** > 0;
options.**nrow_approx** > 0.

crnames – char * Default memory $\mathbf{n} + \mathbf{m}$ array of char *

On exit: the MPSX names of all the variables and constraints in the problem in the following order. **options**.**crnames**$[j-1]$ contains the name of the jth column, for $j = 1, 2, \ldots, \mathbf{n}$. **options**.**crnames**$[\mathbf{n}+i-1]$ contains the name of the ith row, for $i = 1, 2, \ldots, \mathbf{m}$. 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_ip_mps_read (h02buc). The memory freeing function nag_ip_free (h02xzc) should be used to free this memory. You should **not** use the standard C function `free()` for this purpose.

If, on return from nag_ip_mps_read (h02buc), nag_ip_bb (h02bbc) is called with **options** as an argument, and the memory pointed to by **options**.**crnames** has not been freed, nag_ip_bb (h02bbc) will use the row and column names stored in **options**.**crnames** in its solution output.

n_ivar – Integer

On exit: the number of integer variables specified by the data file.

12 Example 2 (ex2)

See Section 10 for the example program.