Module 1.3: nag_write_mat
Matrix Printing

`nag_write_mat` contains procedures for formatted output of matrices.

Contents

Introduction

1 Choice of Procedures

This module contains generic procedures for formatted output of matrices. The available procedures are:

- `nag_write_gen_mat`: for writing a real, complex or integer general matrix.
- `nag_write_tri_mat`: for writing a real or complex triangular matrix; this procedure may also be used for writing the upper or lower triangle of a symmetric or Hermitian matrix.
- `nag_write_bnd_mat`: for writing a real or complex band matrix.

In this document the term 'print' is often used loosely to mean 'write to a formatted file', even though the file may never actually be printed.

2 Storage of Triangular, Symmetric, or Hermitian Matrices

The procedure `nag_write_tri_mat` allows a choice of storage schemes for triangular, symmetric or Hermitian matrices: conventional storage or packed storage. The choice is determined by the rank of the corresponding argument `a`.

2.1 Conventional Storage

The argument `a` is a rank-2 array, of shape (n,n). Matrix element a_{ij} is stored in `a(`i,j`)`. Only the elements of either the upper or the lower triangle need be stored and will be output, as specified by the argument `uplo`; the remaining elements of `a` need not be set.

2.2 Packed Storage

The elements of either the upper or the lower triangle of a square matrix A of order n are packed by columns into contiguous elements of a rank-1 array `a` of shape $(n(n+1)/2)$. The argument `uplo` is used to specify which part of the matrix is packed.

The details of packed storage are as follows.

- If `uplo` = `'u'` or `'U'`, the upper triangle is supplied,
  i.e., a_{ij} is stored in `a(`$i+j(j-1)/2$`)`, for $i \leq j$.
- If `uplo` = `'l'` or `'L'`, the lower triangle is supplied,
  i.e., a_{ij} is stored in `a(`$i+(2n-j)(j-1)/2$`)`, for $i \geq j$.

For example

`uplo`	Square Matrix	Packed storage in array `a`
`'u'` or `'U'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	a_{11}; $\underbrace{a_{12}\,a_{22}}$; $\underbrace{a_{13}\,a_{23}\,a_{33}}$; $\underbrace{a_{14}\,a_{24}\,a_{34}\,a_{44}}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\,a_{21}\,a_{31}\,a_{41}}$; $\underbrace{a_{22}\,a_{32}\,a_{42}}$; $\underbrace{a_{33}\,a_{43}}$; a_{44}

3 Storage of Band Matrices

The procedure `nag_write_bnd_mat` uses the following storage scheme for the square band matrix A with k_l sub-diagonals and k_u super-diagonals:

- a_{ij} must be stored in `a`$(k_u + i - j + 1, j)$, for $\max(j - k_u, 1) \leq i \leq \min(j + k_l, n)$.

For example

Square band matrix A	Band storage in array `a`
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Procedure: nag_write_gen_mat

1 Description

nag_write_gen_mat writes a real, complex or integer general matrix A to a formatted file.

Several optional arguments enable you to control the format in which the matrix is output, but the defaults may well be suitable.

2 Usage

```
USE nag_write_mat
CALL nag_write_gen_mat(a  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array **x** must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

m — the number of rows of the matrix
n — the number of columns of the matrix

3.1 Mandatory Argument

a(m, n) — integer / real(kind=*wp*) / complex(kind=*wp*), intent(in)
Input: the matrix A to be output.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

format — character(len=*), intent(in), optional
Input: specifies the format to be used for printing elements of the matrix A. It must be a valid Fortran format code or one of the special values given below. For complex matrices see also the argument cmplx_form.

A Fortran format code may be any format code allowed on the system, whether it is standard Fortran or not. It need not be enclosed in brackets. Examples of valid values for format are '(F11.4)', '1PE13.5', 'G14.5', for a real or complex matrix, and '(I6)', 'I4,2X' for an integer matrix.

In addition, there are special values which force this procedure to choose its own format.

Real or complex data

If format = ' ', this procedure will choose a format code such that numbers will be printed with either an 'F8.4', an 'F11.4' or a '1PE13.4' format. The 'F8.4' code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The 'F11.4' code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the '1PE13.4' code is chosen.

If format = '*', this procedure will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran 90 compiler in use.

Integer data

If `format = ' '`, this procedure will choose a format code such that numbers will be printed using the smallest field width that is large enough to hold all the numbers to be printed.

Constraints: `format` must be supplied if `cmplx_form` is present. The character length of `format` must be ≤ 80.

Default: `format = ' '`.

cmplx_form — character(len=1), intent(in), optional

Input: this argument is only applicable to complex matrices, and indicates how the value of `format` is to be used to print complex matrix elements.

If `cmplx_form = 'B'` or `'b'` (Bracketed), `format` is assumed to contain a single real edit-descriptor, which is used to print the real and imaginary parts of each complex number separated by a comma, and surrounded by brackets. Complex numbers printed in this format can be read using list-directed input. With `cmplx_form = 'b'` and `format = '(F8.3)'`, a complex number might be printed as `( 12.345, −11.323)`.

If `cmplx_form = 'D'` or `'d'` (Direct), `format` is used unaltered to print a complex number. This `cmplx_form` option allows the user flexibility to specify exactly how the number is printed. With `cmplx_form = 'd'` and `format = '(S, F6.3, SP, F6.3, 'i')'`, a complex number might be printed as `0.123+3.214i`.

If `cmplx_form = 'A'` or `'a'` (Above), `format` is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each complex number one above the other. Each row of the matrix is separated from the next by a blank line, and any row labels are attached only to the real parts. This option means that about twice as many columns can be fitted into `rec_len` characters than if any other `cmplx_form` option is used. A typical value of `format` for this `cmplx_form` option might be `format = 'E13.4'`, `'*'` or `' '`.

Constraints: `cmplx_form` must only be used for complex matrices and must be one of `'A'`, `'a'`, `'B'`, `'b'`, `'D'` or `'d'`.

Default: `cmplx_form = 'B'`.

rec_len — integer, intent(in), optional

Input: the maximum output record length. If the number of columns of the matrix is too large to be accommodated in `rec_len` characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line. `rec_len` must be large enough to hold at least one column of the matrix using the format specifier in `format`, any row labels specified by `int_row_labels` or `row_labels`, and any indentation specified by `indent`.

Constraints: $0 <$ `rec_len` ≤ 132.

Default: `rec_len` $= 80$.

title — character(len=*), intent(in), optional

Input: a title to be printed above the matrix. If `title = ' '`, no title (and no blank line) will be printed. If `title` contains more than `rec_len` characters, the contents of `title` will be wrapped onto more than one line, with the break after `rec_len` characters. Any trailing blank characters in `title` are ignored.

Default: `title = ' '`.

row_labels(*m*) — character(len=*), intent(in), optional

Input: the labels for the rows of the matrix. Labels are right-justified when output, in a field which is as wide as necessary to hold the longest row label.

Default: see `int_row_labels`.

int_row_labels — logical, intent(in), optional

Input: if `row_labels` is not present, then `int_row_labels` indicates the type of labelling to be applied to the rows of the matrix, as follows:

if `int_row_labels` = `.true.`, integer labels (the row numbers);

if `int_row_labels` = `.false.`, no labels.

Note: if `row_labels` is present, `int_row_labels` will be ignored.

Default: `int_row_labels` = `.false.`.

col_labels(m) — character(len=*), intent(in), optional

Input: the labels for the columns of the matrix. Labels are right-justified when output. Any label that is too long for the column width, which is determined by `format`, is truncated.

Default: see `int_col_labels`.

int_col_labels — logical, intent(in), optional

Input: if `col_labels` is not present, then `int_col_labels` indicates the type of labelling to be applied to the columns of the matrix, as follows:

if `int_col_labels` = `.true.`, integer labels (the column numbers);

if `int_col_labels` = `.false.`, no labels.

Note: if `col_labels` is present, `int_col_labels` will be ignored.

Default: `int_col_labels` = `.false.`.

indent — integer, intent(in), optional

Input: the number of columns by which the matrix (and any title and labels) should be indented.

Constraints: $0 \leq$ `indent` $<$ `rec_len`.

Default: `indent` = 0.

unit — integer, intent(in), optional

Input: `unit` specifies the Fortran unit number which identifies the file to be written to.

Constraints: `unit` ≥ 0.

Default: `unit` = the default output unit number for the implementation.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.

Failures (error%level = 2):

error%code	Description
201	Inadequate width for printing a column of the matrix. The quantity `rec_len` − `indent` − w_{L} (where w_{L} is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. One or both of the following may have occurred: both `row_labels` and `int_row_labels` are present, `int_row_labels` will be ignored; both `col_labels` and `int_col_labels` are present, `int_col_labels` will be ignored.
102	The matrix contains no elements. At least one of the dimensions of the matrix `a` is zero.
103	Long column label. At least one of the elements of `col_labels`, after deleting the trailing spaces, is longer than the width of the field allowed to print a column. Any such element will be truncated.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 3 of this module document.

These two examples could be modified to cater for different combinations of the optional arguments and different types of data.

Procedure: nag_write_tri_mat

1 Description

nag_write_tri_mat writes a real or complex triangular matrix A to a formatted file. It allows either conventional or packed storage for A (see the Module Introduction).

Several optional arguments enable you to control the format in which the matrix is output, but the defaults may well be suitable.

Strictly speaking, this procedure outputs the upper or lower triangle of a square matrix. It may therefore be used to output the upper or lower triangle of a symmetric or Hermitian matrix.

2 Usage

```
USE nag_write_mat
CALL nag_write_tri_mat(uplo, a  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the matrix A

The mandatory argument a may have rank 1 or 2, depending on whether packed or conventional storage is used.

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied and is to be output as follows:

if uplo = 'u' or 'U', the upper triangle is supplied and is to be output;

if uplo = 'l' or 'L', the lower triangle is supplied and is to be output.

Constraints: uplo = 'u', 'U', 'l' or 'L'.

a(n, n) / **a**($n(n+1)/2$) — real(kind=*wp*) / complex(kind=*wp*), intent(in)

Input: the matrix A to be output.

Conventional storage (a has shape (n, n))

If uplo = 'u', the upper triangle of A is supplied, and elements below the diagonal need not be set;

if uplo = 'l', the lower triangle of A is supplied, and elements above the diagonal need not be set.

Packed storage (a has shape $(n(n+1)/2)$)

If uplo = 'u', the upper triangle of A is supplied, packed by columns, with a_{ij} in a($i + j(j-1)/2$) for $i \leq j$;

if uplo = 'l', the lower triangle of A is supplied, packed by columns, with a_{ij} in a($i + (2n-j)(j-1)/2$) for $i \geq j$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

diag — character(len=1), intent(in), optional

Input: specifies whether the diagonal elements of the matrix are to be printed, as follows:

if `diag` = `'N'` or `'n'` (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed;

if `diag` = `'B'` or `'b'` (Blank), the diagonal elements of the matrix are not referenced and not printed;

if `diag` = `'U'` or `'u'` (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

Constraints: `diag` = `'B'`, `'b'`, `'U'`, `'u'`, `'N'` or `'n'`.

Default: `diag` = `'N'`.

format — character(len=*), intent(in), optional

Input: specifies the format to be used for printing elements of the matrix A. It must be a valid Fortran format code or one of the special values given below. For complex matrices see also the argument `cmplx_form`.

A Fortran format code may be any format code allowed on the system, whether it is standard Fortran or not. It may or may not be enclosed in brackets. Examples of valid values for `format` are `'(F11.4)'`, `'1PE13.5'`, `'G14.5'`.

In addition, there are special values which force this procedure to choose its own format.

If `format` = `' '`, this procedure will choose a format code such that numbers will be printed with either an `'F8.4'`, an `'F11.4'` or a `'1PE13.4'` format. The `'F8.4'` code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The `'F11.4'` code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the `'1PE13.4'` code is chosen.

If `format` = `'*'`, this procedure will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran 90 compiler in use.

Constraints: `format` must be supplied if `cmplx_form` is present. The character length of `format` must be ≤ 80.

Default: `format` = `' '`.

cmplx_form — character(len=1), intent(in), optional

Input: this argument is only applicable to complex matrices, and indicates how the value of `format` is to be used to print complex matrix elements.

If `cmplx_form` = `'B'` or `'b'` (Bracketed), `format` is assumed to contain a single real edit-descriptor, which is used to print the real and imaginary parts of each complex number separated by a comma, and surrounded by brackets. Complex numbers printed in this format can be read using list-directed input. With `cmplx_form` = `'b'` and `format` = `'(F8.3)'`, a complex number might be printed as `( 12.345, −11.323)`.

If `cmplx_form` = `'D'` or `'d'` (Direct), `format` is used unaltered to print a complex number. This `cmplx_form` option allows the user flexibility to specify exactly how the number is printed. With `cmplx_form` = `'d'` and `format` = `'(S, F6.3, SP, F6.3, 'i')'`, a complex number might be printed as `0.123+3.214i`.

If `cmplx_form` = `'A'` or `'a'` (Above), `format` is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each complex number one above the other. Each row of the matrix is separated from the next by a blank line, and any row labels

are attached only to the real parts. This option means that about twice as many columns can be fitted into rec_len characters than if any other cmplx_form option is used. A typical value of format for this cmplx_form option might be format = 'E13.4', '*' or ' '.

Constraints: cmplx_form must only be used for complex matrices and must be one of 'A', 'a', 'B', 'b', 'D' or 'd'.

Default: cmplx_form = 'B'.

rec_len — integer, intent(in), optional

Input: the maximum output record length. If the number of columns of the matrix is too large to be accommodated in rec_len characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line. rec_len must be large enough to hold at least one column of the matrix using the format specifier in format, any row labels specified by int_row_labels or row_labels, and any indentation specified by indent.

Constraints: $0 <$ rec_len ≤ 132.

Default: rec_len = 80.

title — character(len=*), intent(in), optional

Input: a title to be printed above the matrix. If title = ' ', no title (and no blank line) will be printed. If title contains more than rec_len characters, the contents of title will be wrapped onto more than one line, with the break after rec_len characters. Any trailing blank characters in title are ignored.

Default: title = ' '.

row_labels(*n*) — character(len=*), intent(in), optional

Input: the labels for the rows of the matrix. Labels are right-justified when output, in a field which is as wide as necessary to hold the longest row label.

Default: see int_row_labels.

int_row_labels — logical, intent(in), optional

Input: if row_labels is not present, then int_row_labels indicates the type of labelling to be applied to the rows of the matrix, as follows:

if int_row_labels = .true., integer labels (the row numbers);

if int_row_labels = .false., no labels.

Note: if row_labels is present, int_row_labels will be ignored.

Default: int_row_labels = .false..

col_labels(*n*) — character(len=*), intent(in), optional

Input: the labels for the columns of the matrix. Labels are right-justified when output. Any label that is too long for the column width, which is determined by format, is truncated.

Default: see int_col_labels.

int_col_labels — logical, intent(in), optional

Input: if col_labels is not present, then int_col_labels indicates the type of labelling to be applied to the columns of the matrix, as follows:

if int_col_labels = .true., integer labels (the column numbers);

if int_col_labels = .false., no labels.

Note: if col_labels is present, int_col_labels will be ignored.

Default: int_col_labels = .false..

indent — integer, intent(in), optional

Input: the number of columns by which the matrix (and any title and labels) should be indented.

Constraints: $0 \leq$ `indent` $<$ `rec_len`.

Default: `indent` $= 0$.

unit — integer, intent(in), optional

Input: `unit` specifies the Fortran unit number which identifies the file to be written to.

Constraints: `unit` ≥ 0.

Default: `unit` = the default output unit number for the implementation.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.

Failures (error%level = 2):

error%code	Description
201	Inadequate width for printing a column of the matrix. The quantity `rec_len` $-$ `indent` $- w_{\mathrm{L}}$ (where w_{L} is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. One or both of the following may have occurred: both `row_labels` and `int_row_labels` are present, `int_row_labels` will be ignored; both `col_labels` and `int_col_labels` are present, `int_col_labels` will be ignored.
102	The matrix contains no elements. For a matrix stored in a two-dimensional array, the size along one of the dimensions of `a` is zero. For a packed matrix stored in a one-dimensional array, the size of `a` is zero.
103	Long column label. At least one of the elements of `col_labels`, after deleting the trailing spaces, is longer than the width of the field allowed to print a column. Any such element will be truncated.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

This example could be modified to cater for different combinations of the optional arguments and different types of data.

Procedure: nag_write_bnd_mat

1 Description

nag_write_bnd_mat writes a real or complex square band matrix A stored in a packed two-dimensional array (see the Module Introduction) to a formatted file.

Several optional arguments enable you to control the format in which the matrix is output, but the defaults may well be suitable.

2 Usage

```
USE nag_write_mat
CALL nag_write_bnd_mat(ku, a  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the band matrix A

$k_l \geq 0$ — the number of sub-diagonals in the band matrix A

3.1 Mandatory Arguments

ku — integer, intent(in)

Input: the number k_u of super-diagonals in the band matrix A.

Constraints: ku ≥ 0.

a($k_l + k_u + 1, n$) — real(kind=*wp*) / complex(kind=*wp*), intent(in)

Input: the general band matrix A to be output; a_{ij} must be stored in a($k_u + i - j + 1, j$) for $\max(j - k_u, 1) \leq i \leq \min(j + k_l, n)$.

Note: the diagonal elements of the original matrix are stored in row number ku + 1 of a.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

format — character(len=*), intent(in), optional

Input: specifies the format to be used for printing elements of the matrix A. It must be a valid Fortran format code or one of the special values given below. For complex matrices see also the argument cmplx_form.

A Fortran format code may be any format code allowed on the system, whether it is standard Fortran or not. It may or may not be enclosed in brackets. Examples of valid values for format are '(F11.4)', '1PE13.5', 'G14.5'.

In addition, there are special values which force this procedure to choose its own format.

If format = ' ', this procedure will choose a format code such that numbers will be printed with either an 'F8.4', an 'F11.4' or a '1PE13.4' format. The 'F8.4' code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The 'F11.4' code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the '1PE13.4' code is chosen.

If format = '*', this procedure will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran 90 compiler in use.

Constraints: format must be supplied if cmplx_form is present. The character length of format must be ≤ 80.

Default: format = ' '.

cmplx_form — character(len=1), intent(in), optional

Input: this argument is only applicable to complex matrices, and indicates how the value of format is to be used to print complex matrix elements.

If cmplx_form = 'B' or 'b' (Bracketed), format is assumed to contain a single real edit-descriptor, which is used to print the real and imaginary parts of each complex number separated by a comma, and surrounded by brackets. Complex numbers printed in this format can be read using list-directed input. With cmplx_form = 'b' and format = '(F8.3)', a complex number might be printed as (12.345, −11.323).

If cmplx_form = 'D' or 'd' (Direct), format is used unaltered to print a complex number. This cmplx_form option allows the user flexibility to specify exactly how the number is printed. With cmplx_form = 'd' and format = '(S, F6.3, SP, F6.3, 'i')', a complex number might be printed as 0.123+3.214i.

If cmplx_form = 'A' or 'a' (Above), format is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each complex number one above the other. Each row of the matrix is separated from the next by a blank line, and any row labels are attached only to the real parts. This option means that about twice as many columns can be fitted into rec_len characters than if any other cmplx_form option is used. A typical value of format for this cmplx_form option might be format = 'E13.4', '*' or ' '.

Constraints: cmplx_form must only be used for complex matrices and must be one of 'A', 'a', 'B', 'b', 'D' or 'd'.

Default: cmplx_form = 'B'.

rec_len — integer, intent(in), optional

Input: the maximum output record length. If the number of columns of the matrix is too large to be accommodated in rec_len characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line. rec_len must be large enough to hold at least one column of the matrix using the format specifier in format, any row labels specified by int_row_labels or row_labels, and any indentation specified by indent.

Constraints: $0 <$ rec_len ≤ 132.

Default: rec_len = 80.

title — character(len=*), intent(in), optional

Input: a title to be printed above the matrix. If title = ' ', no title (and no blank line) will be printed. If title contains more than rec_len characters, the contents of title will be wrapped onto more than one line, with the break after rec_len characters. Any trailing blank characters in title are ignored.

Default: title = ' '.

row_labels(*n*) — character(len=*), intent(in), optional

Input: the labels for the rows of the matrix. Labels are right-justified when output, in a field which is as wide as necessary to hold the longest row label.

Default: see `int_row_labels`.

int_row_labels — logical, intent(in), optional

Input: if `row_labels` is not present, then `int_row_labels` indicates the type of labelling to be applied to the rows of the matrix, as follows:

if `int_row_labels` = `.true.`, integer labels (the row numbers);

if `int_row_labels` = `.false.`, no labels.

Note: if `row_labels` is present, `int_row_labels` will be ignored.

Default: `int_row_labels` = `.false.`.

col_labels(*n*) — character(len=*), intent(in), optional

Input: the labels for the columns of the matrix. Labels are right-justified when output. Any label that is too long for the column width, which is determined by `format`, is truncated.

Default: see `int_col_labels`.

int_col_labels — logical, intent(in), optional

Input: if `col_labels` is not present, then `int_col_labels` indicates the type of labelling to be applied to the columns of the matrix, as follows:

if `int_col_labels` = `.true.`, integer labels (the column numbers);

if `int_col_labels` = `.false.`, no labels.

Note: if `col_labels` is present, `int_col_labels` will be ignored.

Default: `int_col_labels` = `.false.`.

indent — integer, intent(in), optional

Input: the number of columns by which the matrix (and any title and labels) should be indented.

Constraints: $0 \leq$ `indent` $<$ `rec_len`.

Default: `indent` = 0.

unit — integer, intent(in), optional

Input: `unit` specifies the Fortran unit number which identifies the file to be written to.

Constraints: `unit` ≥ 0.

Default: `unit` = the default output unit number for the implementation.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.

Failures (error%level = 2):

error%code	Description
201	Inadequate width for printing a column of the matrix. The quantity `rec_len` − `indent` − w_{L} (where w_{L} is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

Warnings (error%level = 1):

error%code	Description
101	Optional argument present but not used. One or both of the following may have occurred: both `row_labels` and `int_row_labels` are present, `int_row_labels` will be ignored; both `col_labels` and `int_col_labels` are present, `int_col_labels` will be ignored.
102	The matrix contains no elements. The order of the matrix A (the second dimension of `a`) is zero.
103	Long column label. At least one of the elements of `col_labels`, after deleting the trailing spaces, is longer than the width of the field allowed to print a column. Any such element will be truncated.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 4 of this module document.

This example could be modified to cater for different combinations of the optional arguments and different types of data.

Example 1: Writing a Real General Matrix

This example program calls `nag_write_gen_mat` to write a real matrix using the default values for the optional arguments. It then rewrites the same matrix after supplying some of the optional arguments to produce title, integer labels and different format.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_write_mat_ex01

  ! Example Program Text for nag_write_mat
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 4, n = 3
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j
  CHARACTER (80) :: title
  ! .. Local Arrays ..
  REAL (wp) :: a(m,n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_write_mat_ex01'

  ! Generate an array of data

  DO j = 1, n
    DO i = 1, m
      a(i,j) = REAL(10*i+j,kind=wp)
    END DO
  END DO

  ! Write rectangular matrix with no optional parameters set

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(a)

  ! Write the same matrix with title, format and integer labels

  title = 'Output of the same matrix with title and integer &
   &labels using F8.1 format'

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(a,format='F8.1',title=title, &
   int_row_labels=.TRUE.,int_col_labels=.TRUE.)

END PROGRAM nag_write_mat_ex01
```

2 Program Data

None.

3 Program Results

```
Example Program Results for nag_write_mat_ex01

     11.0000    12.0000    13.0000
     21.0000    22.0000    23.0000
     31.0000    32.0000    33.0000
     41.0000    42.0000    43.0000

Output of the same matrix with title and integer labels using F8.1 format
         1       2       3
1     11.0    12.0    13.0
2     21.0    22.0    23.0
3     31.0    32.0    33.0
4     41.0    42.0    43.0
```

Example 2: Writing a Real Triangular Matrix

This example program calls `nag_write_tri_mat` to write a real upper triangle matrix supplied in conventional storage using the default setting for the optional arguments. It then writes a real lower triangle matrix supplied in packed storage using the supplied title and unit diagonal elements.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_write_mat_ex02

  ! Example Program Text for nag_write_mat
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_write_mat, ONLY : nag_write_tri_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 5, n = 4
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j
  CHARACTER (160) :: title
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp) :: a(n,n), p((m*(m+1))/2)
  CHARACTER (6) :: row_labels(m)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_write_mat_ex02'

  ! Generate an array of data

  DO i = 1, n
    DO j = i, n
      a(i,j) = REAL(10*i+j,kind=wp)
    END DO
  END DO

  ! Write real upper triangular matrix stored in convential storage
  ! using default optional arguments

  uplo = 'U'
  WRITE (nag_std_out,*)

  CALL nag_write_tri_mat(uplo,a)

  ! Generate arrays of data and row labels

  DO i = 1, (m*(m+1))/2
    p(i) = REAL(10*i,kind=wp)
  END DO

  DO i = 1, m
    WRITE (row_labels(i),'(a,i3)') 'row', i
  END DO
```

```
      ! Write real lower triangular matrix stored in packed storage
      ! using title, unit diagonal elements, character row labels
      ! and integer column labels

      uplo = 'l'
      title = 'Output real lower triangular matrix in packed storage &
       &with unit diagonal        elements, title, character row &
       &labels and integer column labels'

      WRITE (nag_std_out,*)

      CALL nag_write_tri_mat(uplo,p,diag='u',title=title, &
       row_labels=row_labels,int_col_labels=.TRUE.)

    END PROGRAM nag_write_mat_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_write_mat_ex02

      11.0000    12.0000    13.0000    14.0000
                 22.0000    23.0000    24.0000
                            33.0000    34.0000
                                       44.0000

 Output real lower triangular matrix in packed storage with unit diagonal
  elements, title, character row labels and integer column labels
                    1          2          3          4          5
 row  1      1.0000
 row  2     20.0000     1.0000
 row  3     30.0000    70.0000     1.0000
 row  4     40.0000    80.0000   110.0000     1.0000
 row  5     50.0000    90.0000   120.0000   140.0000     1.0000
```

Example 3: Writing a Complex General Matrix

This example program calls `nag_write_gen_mat` to write a complex matrix in three different styles as follows:

using the default setting for all the optional arguments;

using integer labels, the supplied title and setting `cmplx_form = 'a'`;

using the supplied title and format, and setting `cmplx_form = 'd'`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_write_mat_ex03

  ! Example Program Text for nag_write_mat
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC CMPLX, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 4, n = 3
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, k, l
  CHARACTER (30) :: format
  CHARACTER (80) :: title
  ! .. Local Arrays ..
  COMPLEX (wp) :: a(m,n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_write_mat_ex03'

  ! Generate an array of data

  l = -1
  DO j = 1, n
    l = -l*j
    DO i = 1, m
      k = (-1)**(i+j)
      a(i,j) = CMPLX(l,k*(i+j),kind=wp)
    END DO
  END DO

  ! Write a complex matrix using defaults

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(a)

  ! Write a complex matrix with title, integer labels and
  ! cmplx_form='a'

  title = &
   'Output the same matrix with title, integer labels and cmplx_form="a"'

  WRITE (nag_std_out,*)
```

```
      CALL nag_write_gen_mat(a,title=title,int_row_labels=.TRUE., &
       int_col_labels=.TRUE.,cmplx_form='a')

      ! Write a complex matrix with title, integer labels and
      ! cmplx_form='d'

      title = 'Output the same matrix with title and cmplx_form="d"'
      format = '(s,f7.2,sp,f5.2,"i")'
      WRITE (nag_std_out,*)

      CALL nag_write_gen_mat(a,title=title,format=format,cmplx_form='d')

    END PROGRAM nag_write_mat_ex03
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_write_mat_ex03

   (     1.0000,     2.0000) (    -2.0000,    -3.0000) (     6.0000,     4.0000)
   (     1.0000,    -3.0000) (    -2.0000,     4.0000) (     6.0000,    -5.0000)
   (     1.0000,     4.0000) (    -2.0000,    -5.0000) (     6.0000,     6.0000)
   (     1.0000,    -5.0000) (    -2.0000,     6.0000) (     6.0000,    -7.0000)

 Output the same matrix with title, integer labels and cmplx_form="a"
             1          2          3
 1      1.0000    -2.0000     6.0000
        2.0000    -3.0000     4.0000

 2      1.0000    -2.0000     6.0000
       -3.0000     4.0000    -5.0000

 3      1.0000    -2.0000     6.0000
        4.0000    -5.0000     6.0000

 4      1.0000    -2.0000     6.0000
       -5.0000     6.0000    -7.0000

 Output the same matrix with title and cmplx_form="d"
     1.00+2.00i  -2.00-3.00i   6.00+4.00i
     1.00-3.00i  -2.00+4.00i   6.00-5.00i
     1.00+4.00i  -2.00-5.00i   6.00+6.00i
     1.00-5.00i  -2.00+6.00i   6.00-7.00i
```

Example 4: Writing a Real Band Matrix

This example program calls `nag_write_bnd_mat` to write a real band matrix using the default values for the optional arguments. It then rewrites sections of the same matrix after supplying some of the optional arguments.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_write_mat_ex04

  ! Example Program Text for nag_write_mat
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_write_mat, ONLY : nag_write_bnd_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: kl = 1, ku = 2, n = 5
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  INTEGER, PARAMETER :: diag_index = ku + 1
  ! .. Local Scalars ..
  INTEGER :: i, j
  CHARACTER (80) :: title
  ! .. Local Arrays ..
  REAL (wp) :: a(kl+ku+1,n)
  CHARACTER (6) :: col_labels(n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_write_mat_ex04'

  ! Generate an array of data

  DO i = 1, kl + ku + 1
    DO j = 1, n
      a(i,j) = REAL(10*i+j,kind=wp)
    END DO
  END DO

  ! Write the complex 5 by 5 banded matrix stored in a

  WRITE (nag_std_out,*)

  CALL nag_write_bnd_mat(ku,a)

  ! Write the upper part of the 4 by 4 section of the banded matrix
  ! stored in a, using a title

  title = 'Output the upper part of a section of the band matrix'
  WRITE (nag_std_out,*)

  CALL nag_write_bnd_mat(ku,a(1:diag_index,1:4),title=title)

  ! Generate column labels

  DO i = 1, n
    WRITE (col_labels(i),'(a,i3)') 'col', i
```

```
      END DO

      ! Write one sub-diagonal and one super diagonal of the banded
      ! matrix stored in a, using a title, character column labels
      ! and integer row labels

      title = &
       'Output the band matrix with one sub-diagonal and one super-diagonal'
      WRITE (nag_std_out,*)

      CALL nag_write_bnd_mat(1,a(diag_index-1:diag_index+1,:),title=title, &
       col_labels=col_labels,int_row_labels=.TRUE.)

    END PROGRAM nag_write_mat_ex04
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_write_mat_ex04

      31.0000    22.0000    13.0000
      41.0000    32.0000    23.0000    14.0000
                 42.0000    33.0000    24.0000    15.0000
                            43.0000    34.0000    25.0000
                                       44.0000    35.0000

 Output the upper part of a section of the band matrix
      31.0000    22.0000    13.0000
                 32.0000    23.0000    14.0000
                            33.0000    24.0000
                                       34.0000

 Output the band matrix with one sub-diagonal and one super-diagonal
        col  1     col  2     col  3     col  4     col  5
 1     31.0000    22.0000
 2     41.0000    32.0000    23.0000
 3                42.0000    33.0000    24.0000
 4                           43.0000    34.0000    25.0000
 5                                      44.0000    35.0000
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_write_mat_ex05`
: Writing a real general matrix with specified row labels.

`nag_write_mat_ex06`
: Writing the lower triangular part, without diagonal elements, of a complex general matrix.

`nag_write_mat_ex07`
: Writing an integer general matrix with specified row and column labels.

`nag_write_mat_ex08`
: Writing the upper triangular part of a complex triangular matrix stored with packed storage.

`nag_write_mat_ex09`
: Writing a real lower triangular matrix stored with packed storage.

`nag_write_mat_ex10`
: Writing a complex band matrix.