NAG Library Function Document

nag_gen_complx_mat_print (x04dac)

1 Purpose

nag_gen_complx_mat_print (x04dac) is an easy-to-use function to print a Complex matrix.

2 Specification

```
#include <nag.h>
#include <nagx04.h>
void nag_gen_complx_mat_print (Nag_OrderType order, Nag_MatrixType matrix,
     Nag_DiagType diag, Integer m, Integer n, const Complex a[], Integer pda,
     const char *title, const char *outfile, NagError *fail)
```

3 Description

nag_gen_complx_mat_print (x04dac) prints a Complex matrix. It is an easy-to-use driver for nag_gen_complx_mat_print_comp (x04dbc). The function uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

nag_gen_complx_mat_print (x04dac) will choose a format code such that numbers will be printed with a %8.4f, a %11.4f or a %13.4e format. The %8.4f code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The %11.4f code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the %13.4e code is chosen. The chosen code is used to print each complex element of the matrix with the real part above the imaginary part.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the file specified by **outfile** or, by default, to standard output.

4 References

None.

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **matrix** – Nag_MatrixType *Input*

On entry: indicates the part of the matrix to be printed.

matrix = Nag_GeneralMatrix
The whole of the rectangular matrix.

matrix = Nag_LowerMatrix
The lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

matrix = Nag_UpperMatrix
The upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

Constraint: **matrix** = Nag_GeneralMatrix, Nag_LowerMatrix or Nag_UpperMatrix.

3: **diag** – Nag_DiagType *Input*

On entry: indicates whether the diagonal elements of the matrix are to be printed.

diag = Nag_NonRefDiag
The diagonal elements of the matrix are not referenced and not printed.

diag = Nag_UnitDiag
The diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

diag = Nag_NonUnitDiag
The diagonal elements of the matrix are referenced and printed.

If **matrix** = Nag_GeneralMatrix, then **diag** must be set to Nag_NonUnitDiag.

Constraints:

if **matrix** $\neq$ Nag_GeneralMatrix, **diag** = Nag_NonRefDiag, Nag_UnitDiag or Nag_NonUnitDiag;
if **matrix** = Nag_GeneralMatrix, **diag** = Nag_NonUnitDiag.

4: **m** – Integer *Input*
5: **n** – Integer *Input*

On entry: the number of rows and columns of the matrix, respectively, to be printed.

If either **m** or **n** is less than 1, nag_gen_complx_mat_print (x04dac) will exit immediately after printing **title**; no row or column labels are printed.

6: **a**$[dim]$ – const Complex *Input*

Note: the dimension, *dim*, of the array **a** must be at least

$\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix A is stored in

a$[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
a$[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the matrix to be printed. Only the elements that will be referred to, as specified by arguments **matrix** and **diag**, need be set.

7: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraints:

if **order** = Nag_ColMajor, **pda** $\geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, **pda** $\geq \max(1, \mathbf{n})$.

8: **title** – const char * *Input*

On entry: a title to be printed above the matrix.

If **title** = **NULL**, no title (and no blank line) will be printed.

If **title** contains more than 80 characters, the contents of **title** will be wrapped onto more than one line, with the break after 80 characters.

Any trailing blank characters in **title** are ignored.

9: **outfile** – const char * *Input*

On entry: the name of a file to which output will be directed. If **outfile** is **NULL** the output will be directed to standard output.

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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_NOT_APPEND_FILE

Cannot open file ⟨*value*⟩ for appending.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*value*⟩.

NE_NOT_WRITE_FILE

Cannot open file ⟨*value*⟩ for writing.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_gen_complx_mat_print (x04dac) is not threaded in any implementation.

9 Further Comments

A call to nag_gen_complx_mat_print (x04dac) is equivalent to a call to nag_gen_complx_mat_print_comp (x04dbc) with the following argument values:

```
ncols = 80
 indent = 0
 labrow = Nag_IntegerLabels
 labcol = Nag_IntegerLabels
 form = 0
 cmplxform = Nag_AboveForm
```

10 Example

This example program calls nag_gen_complx_mat_print (x04dac) twice, first to print a 4 by 3 rectangular matrix, and then to print a 4 by 4 lower triangular matrix.

10.1 Program Text

```
/* nag_gen_complx_mat_print (x04dac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf01.h>
#include <nagx04.h>

int main(void)
{
  Integer       exit_status = 0, n = 5;
  Integer       i, j, pda;
  double        aa;
  Complex       *a = 0;
  /* Nag Types */
  Nag_OrderType order;
  NagError      fail;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I-1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J-1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_gen_complx_mat_print (x04dac) Example Program Results\n\n");

  pda = n;
  if (!(a = NAG_ALLOC(pda * n, Complex))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Generate a full-format symmetric array of data. */
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      aa = (double) (10 * i + j);
      A(i, j) = nag_complex(aa, -aa);
    }
  }

  /* Print n by (n-1) rectangular matrix
   * using nag_geb_complx_mat_print (x04dac).
   */
  fflush(stdout);
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n - 1,
                           a, pda, "Example 1:", NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
           fail.message);
    exit_status = 1;
```

```
    goto END;
  }
  printf("\n");

  /* Print the order-n upper-triangular matrix with Unit diagonal
   * using nag_gen_complx_mat_print (x04dac).
   */
  fflush(stdout);
  nag_gen_complx_mat_print(order, Nag_UpperMatrix, Nag_UnitDiag, n, n,
                           a, pda, "Example 2:", NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }
  printf("\n");

END:
  NAG_FREE(a);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_gen_complx_mat_print (x04dac) Example Program Results

 Example 1:
             1           2           3           4
 1     11.0000     12.0000     13.0000     14.0000
      -11.0000    -12.0000    -13.0000    -14.0000

 2     21.0000     22.0000     23.0000     24.0000
      -21.0000    -22.0000    -23.0000    -24.0000

 3     31.0000     32.0000     33.0000     34.0000
      -31.0000    -32.0000    -33.0000    -34.0000

 4     41.0000     42.0000     43.0000     44.0000
      -41.0000    -42.0000    -43.0000    -44.0000

 5     51.0000     52.0000     53.0000     54.0000
      -51.0000    -52.0000    -53.0000    -54.0000


 Example 2:
             1           2           3           4           5
 1      1.0000     12.0000     13.0000     14.0000     15.0000
        0.0000    -12.0000    -13.0000    -14.0000    -15.0000

 2                 1.0000     23.0000     24.0000     25.0000
                   0.0000    -23.0000    -24.0000    -25.0000

 3                             1.0000     34.0000     35.0000
                               0.0000    -34.0000    -35.0000

 4                                         1.0000     45.0000
                                           0.0000    -45.0000

 5                                                     1.0000
                                                       0.0000
```