NAG Library Function Document

nag_matop_complex_gen_matrix_log (f01fjc)

1 Purpose

nag_matop_complex_gen_matrix_log (f01fjc) computes the principal matrix logarithm, $\log(A)$, of a complex n by n matrix A, with no eigenvalues on the closed negative real line.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_complex_gen_matrix_log (Nag_OrderType order, Integer n,
     Complex a[], Integer pda, NagError *fail)
```

3 Description

Any nonsingular matrix A has infinitely many logarithms. For a matrix with no eigenvalues on the closed negative real line, the principal logarithm is the unique logarithm whose spectrum lies in the strip $\{z : -\pi < \mathrm{Im}(z) < \pi\}$. If A is nonsingular but has eigenvalues on the negative real line, the principal logarithm is not defined, but nag_matop_complex_gen_matrix_log (f01fjc) will return a non-principal logarithm.

$\log(A)$ is computed using the inverse scaling and squaring algorithm for the matrix logarithm described in Al–Mohy and Higham (2011).

4 References

Al–Mohy A H and Higham N J (2011) Improved inverse scaling and squaring algorithms for the matrix logarithm *SIAM J. Sci. Comput.* **34(4)** C152–C169

Higham N J (2008) *Functions of Matrices: Theory and Computation* SIAM, Philadelphia, PA, USA

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: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 0.

3: **a**[dim] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least **pda** $\times$ **n**.

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

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

On entry: the n by n matrix A.

On exit: the n by n principal matrix logarithm, $\log(A)$, unless **fail.code** = NE_EIGENVALUES, in which case a non-principal logarithm is returned.

4: **pda** – Integer *Input*

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

Constraint: **pda** $\geq$ **n**.

5: **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

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_EIGENVALUES

A was found to have eigenvalues on the negative real line. The principal logarithm is not defined in this case, so a non-principal logarithm was returned.

NE_INT

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

NE_INT_2

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** $\geq$ **n**.

NE_INTERNAL_ERROR

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

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_SINGULAR

A is singular so the logarithm cannot be computed.

NW_SOME_PRECISION_LOSS

$\log(A)$ has been computed using an IEEE double precision Padé approximant, although the arithmetic precision is higher than IEEE double precision.

7 Accuracy

For a normal matrix A (for which $A^{\mathrm{H}}A = AA^{\mathrm{H}}$), the Schur decomposition is diagonal and the algorithm reduces to evaluating the logarithm of the eigenvalues of A and then constructing $\log(A)$ using the Schur vectors. This should give a very accurate result. In general, however, no error bounds are available for the algorithm. See Al–Mohy and Higham (2011) and Section 9.4 of Higham (2008) for details and further discussion.

The sensitivity of the computation of $\log(A)$ is worst when A has an eigenvalue of very small modulus or has a complex conjugate pair of eigenvalues lying close to the negative real axis.

If estimates of the condition number of the matrix logarithm are required then nag_matop_complex_gen_matrix_cond_log (f01kjc) should be used.

8 Parallelism and Performance

nag_matop_complex_gen_matrix_log (f01fjc) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_matop_complex_gen_matrix_log (f01fjc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The cost of the algorithm is $O(n^3)$ floating-point operations (see Al–Mohy and Higham (2011)). The Complex allocatable memory required is approximately $3 \times n^2$.

If the Fréchet derivative of the matrix logarithm is required then nag_matop_complex_gen_matrix_frcht_log (f01kkc) should be used.

nag_matop_real_gen_matrix_log (f01ejc) can be used to find the principal logarithm of a real matrix.

10 Example

This example finds the principal matrix logarithm of the matrix

$$A = \begin{pmatrix} 1.0+2.0i & 0.0+1.0i & 1.0+0.0i & 3.0+2.0i \\ 0.0+3.0i & -2.0+0.0i & 0.0+0.0i & 1.0+0.0i \\ 1.0+0.0i & -2.0+0.0i & 3.0+2.0i & 0.0+3.0i \\ 2.0+0.0i & 0.0+1.0i & 0.0+1.0i & 2.0+3.0i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_log (f01fjc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{

  /* Scalars */
```

```
  Integer exit_status = 0;
  Integer i, j, n, pda;

  /* Arrays */
  Complex *a = 0;

  /* Nag Types */
  Nag_OrderType order;
  NagError fail;

  INIT_FAIL(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

  /* Output preamble */
  printf("nag_matop_complex_gen_matrix_log (f01fjc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read in the problem size */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &n);
#endif

  pda = n;

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

  /* Read in the matrix a from data file */
  for (i = 1; i <= n; i++)
#ifdef _WIN32
    for (j = 1; j <= n; j++)
      scanf_s(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
#else
    for (j = 1; j <= n; j++)
      scanf(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Find log(A) using
   * nag_matop_complex_gen_matrix_log (f01fjc)
   * Complex matrix logarithm
   */
  nag_matop_complex_gen_matrix_log(order, n, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_complex_gen_matrix_log (f01fjc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print solution using
   * nag_gen_complx_mat_print (x04dac)
   * Print complex general matrix (easy to use)
```

```
   */
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           n, n, a, n, "log(A)", 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;
  }

END:
  NAG_FREE(a);
  return exit_status;
}
```

10.2 Program Data

```
nag_matop_complex_gen_matrix_log (f01fjc) Example Program Data
 4                                          :Value of n
(1.0, 2.0) ( 0.0, 1.0) (1.0, 0.0) (3.0, 2.0)
(0.0, 3.0) (-2.0, 0.0) (0.0, 0.0) (1.0, 0.0)
(1.0, 0.0) (-2.0, 0.0) (3.0, 2.0) (0.0, 3.0)
(2.0, 0.0) ( 0.0, 1.0) (0.0, 1.0) (2.0, 3.0) :End of matrix a
```

10.3 Program Results

```
nag_matop_complex_gen_matrix_log (f01fjc) Example Program Results

 log(A)
              1           2           3           4
 1      1.0390      0.2859      0.0516      0.7586
        1.1672      0.3998     -0.2562     -0.4678

 2     -2.7481      1.1898      0.1369      2.1771
        2.6187     -2.2287     -0.9128     -1.0118

 3     -0.8514     -0.2517      1.3839      1.1920
        0.3927     -0.4791      0.2129      0.4240

 4      1.1970     -0.6813      0.0051      0.7867
       -0.1242      0.3969      0.3511      0.7502
```