NAG Library Function Document

nag_matop_complex_gen_matrix_exp (f01fcc)

1 Purpose

nag_matop_complex_gen_matrix_exp (f01fcc) computes the matrix exponential, e^A, of a complex n by n matrix A.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
```

```
void nag_matop_complex_gen_matrix_exp (Nag_OrderType order, Integer n,
     Complex a[], Integer pda, NagError *fail)
```

3 Description

e^A is computed using a Padé approximant and the scaling and squaring method described in Al–Mohy and Higham (2009).

4 References

Al–Mohy A H and Higham N J (2009) A new scaling and squaring algorithm for the matrix exponential *SIAM J. Matrix Anal.* **31(3)** 970–989

Higham N J (2005) The scaling and squaring method for the matrix exponential revisited *SIAM J. Matrix Anal. Appl.* **26(4)** 1179–1193

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

Moler C B and Van Loan C F (2003) Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later *SIAM Rev.* **45** 3–49

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 3.2.1.3 in the Essential Introduction 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 matrix exponential e^A.

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 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

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

NE_BAD_PARAM

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

NE_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 the Essential Introduction 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 the Essential Introduction for further information.

NE_SINGULAR

The linear equations to be solved are nearly singular and the Padé approximant probably has no correct figures; it is likely that this function has been called incorrectly.

The linear equations to be solved for the Padé approximant are singular; it is likely that this function has been called incorrectly.

NW_SOME_PRECISION_LOSS

e^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 computed matrix, e^A, is guaranteed to be close to the exact matrix, that is, the method is forward stable. No such guarantee can be given for non-normal matrices. See Al–Mohy and Higham (2009) and Section 10.3 of Higham (2008) for details and further discussion.

If estimates of the condition number of the matrix exponential are required then nag_matop_complex_gen_matrix_cond_exp (f01kgc) should be used.

8 Parallelism and Performance

nag_matop_complex_gen_matrix_exp (f01fcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_complex_gen_matrix_exp (f01fcc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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

9 Further Comments

The Integer allocatable memory required is **n**, and the Complex allocatable memory required is approximately 6 × **n**^{2}.

The cost of the algorithm is $O(n^3)$; see Section 5 of Al–Mohy and Higham (2009). The complex allocatable memory required is approximately $6 \times n^2$.

If the Fréchet derivative of the matrix exponential is required then nag_matop_complex_gen_matrix_frcht_exp (f01khc) should be used.

As well as the excellent book cited above, the classic reference for the computation of the matrix exponential is Moler and Van Loan (2003).

10 Example

This example finds the matrix exponential of the matrix

$$A = \begin{pmatrix} 1+i & 2+i & 2+i & 2+i \\ 3+2i & 1 & 1 & 2+i \\ 3+2i & 2+i & 1 & 2+i \\ 3+2i & 3+2i & 3+2i & 1+i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_exp (f01fcc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
```

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

  /* Arrays */
  Complex       *a = 0;

  /* NAG types */
  Nag_OrderType order;
  NagError      fail;

  INIT_FAIL(fail);

  printf("nag_matop_complex_gen_matrix_exp (f01fcc) Example Program Results");
  printf("\n\n");
  fflush(stdout);

  /* Read matrix dimension from data file. */
#ifdef _WIN32
  scanf_s("%*[^\n]%"NAG_IFMT"%*[^\n]", &n);
#else
  scanf("%*[^\n]%"NAG_IFMT"%*[^\n]", &n);
#endif

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

#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

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

  /* nag_matop_complex_gen_matrix_exp (f01fcc) - Complex matrix exponential */
  nag_matop_complex_gen_matrix_exp(order, n, a, pda, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_gen_complx_mat_print (x04dac) - Print complex general matrix */
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           n, n, a, n, "Exp(A)", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("%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_exp (f01fcc) Example Program Data

  4                                            :Value of N

  (1.0,1.0)  (2.0,1.0)  (2.0,1.0)  (2.0,1.0)
  (3.0,2.0)  (1.0,0.0)  (1.0,0.0)  (2.0,1.0)
  (3.0,2.0)  (2.0,1.0)  (1.0,0.0)  (2.0,1.0)
  (3.0,2.0)  (3.0,2.0)  (3.0,2.0)  (1.0,1.0)  :End of matrix A
```

10.3 Program Results

```
nag_matop_complex_gen_matrix_exp (f01fcc) Example Program Results

 Exp(A)
             1           2           3           4
 1   -157.9003   -194.6526   -186.5627   -155.7669
     -754.3717   -555.0507   -475.4533   -520.1876

 2   -206.8899   -225.4985   -212.4414   -186.5627
     -694.7443   -505.3938   -431.0611   -475.4533

 3   -208.7476   -238.4962   -225.4985   -194.6526
     -808.2090   -590.8045   -505.3938   -555.0507

 4   -133.3958   -208.7476   -206.8899   -157.9003
    -1085.5496   -808.2090   -694.7443   -754.3717
```