NAG Library Function Document

nag_matop_complex_gen_matrix_actexp (f01hac)

1 Purpose

nag_matop_complex_gen_matrix_actexp (f01hac) computes the action of the matrix exponential e^{tA}, on the matrix B, where A is a complex n by n matrix, B is a complex n by m matrix and t is a complex scalar.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_complex_gen_matrix_actexp (Integer n, Integer m, Complex a[],
     Integer pda, Complex b[], Integer pdb, Complex t, NagError *fail)
```

3 Description

$e^{tA}B$ is computed using the algorithm described in Al–Mohy and Higham (2011) which uses a truncated Taylor series to compute the product $e^{tA}B$ without explicitly forming e^{tA}.

4 References

Al–Mohy A H and Higham N J (2011) Computing the action of the matrix exponential, with an application to exponential integrators *SIAM J. Sci. Statist. Comput.* **33(2)** 488-511

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

5 Arguments

1: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

2: **m** – Integer *Input*

On entry: m, the number of columns of the matrix B.

Constraint: $\mathbf{m} \geq 0$.

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 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

On entry: the n by n matrix A.

On exit: A is overwritten during the computation.

4: **pda** – Integer *Input*

On entry: the stride separating matrix row elements in the array **a**.

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

5: **b**[dim] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least **pdb** × **m**.

The (i, j)th element of the matrix B is stored in **b**[$(j-1) \times$ **pdb** $+ i - 1$].

On entry: the n by m matrix B.

On exit: the n by m matrix $e^{tA}B$.

6: **pdb** – Integer *Input*

On entry: the stride separating matrix row elements in the array **b**.

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

7: **t** – Complex *Input*

On entry: the scalar t.

8: **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 2.3.1.2 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_INT

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

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

NE_INT_2

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

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdb** $\geq$ **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 2.7.6 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 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NW_SOME_PRECISION_LOSS

$e^{tA}B$ has been computed using an IEEE double precision Taylor series, although the arithmetic precision is higher than IEEE double precision.

7 Accuracy

For a Hermitian matrix A (for which $A^{\mathrm{H}} = A$) the computed matrix $e^{tA}B$ is guaranteed to be close to the exact matrix, that is, the method is forward stable. No such guarantee can be given for non-Hermitian matrices. See Section 4 of Al–Mohy and Higham (2011) for details and further discussion.

8 Parallelism and Performance

nag_matop_complex_gen_matrix_actexp (f01hac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_complex_gen_matrix_actexp (f01hac) 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 matrix $e^{tA}B$ could be computed by explicitly forming e^{tA} using nag_matop_complex_gen_matrix_exp (f01fcc) and multiplying B by the result. However, experiments show that it is usually both more accurate and quicker to use nag_matop_complex_gen_matrix_actexp (f01hac).

The cost of the algorithm is $O\left(n^2m\right)$. The precise cost depends on A since a combination of balancing, shifting and scaling is used prior to the Taylor series evaluation.

Approximately $n^2 + (2m + 8)n$ of complex allocatable memory is required by nag_matop_complex_gen_matrix_actexp (f01hac).

nag_matop_real_gen_matrix_actexp (f01gac) can be used to compute $e^{tA}B$ for real A, B, and t. nag_matop_complex_gen_matrix_actexp_rcomm (f01hbc) provides an implementation of the algorithm with a reverse communication interface, which returns control to the user when matrix multiplications are required. This should be used if A is large and sparse.

10 Example

This example computes $e^{tA}B$, where

$$A = \begin{pmatrix} 0.5 + 0.0i & -0.2 + 0.0i & 1.0 + 0.1i & 0.0 + 0.4i \\ 0.3 + 0.0i & 0.5 + 1.2i & 3.1 + 0.0i & 1.0 + 0.2i \\ 0.0 + 2.0i & 0.1 + 0.0i & 1.2 + 0.2i & 0.5 + 0.0i \\ 1.0 + 0.3i & 0.0 + 0.2i & 0.0 + 0.9i & 0.5 + 0.0i \end{pmatrix},$$

$$B = \begin{pmatrix} 0.4 + 0.0i & 1.2 + 0.0i \\ 1.3 + 0.0i & -0.2 + 0.1i \\ 0.0 + 0.3i & 2.1 + 0.0i \\ 0.4 + 0.0i & -0.9 + 0.0i \end{pmatrix}$$

and

$$t = -0.5 + 0.0i.$$

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_actexp (f01hac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{

  /* Scalars */
  Integer exit_status = 0;
  Integer i, j, m, n, lda, ldb;
  Complex t;

  /* Arrays */
  Complex *a = 0;
  Complex *b = 0;

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

  INIT_FAIL(fail);

#define A(I, J) a[(J-1)*lda + I-1]
#define B(I, J) b[(J-1)*ldb + I-1]

  order = Nag_ColMajor;

  /* Output preamble */
  printf("nag_matop_complex_gen_matrix_actexp (f01hac) ");
  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 and the value of the parameter t */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT " %" NAG_IFMT " (%lf , %lf ) %*[^\n]", &n, &m, &t.re,
          &t.im);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT " (%lf , %lf ) %*[^\n]", &n, &m, &t.re,
        &t.im);
#endif

  lda = n;
  ldb = n;

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

```
  /* Read in the matrix 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

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

  /* Find exp(tA) B using
   * nag_matop_complex_gen_matrix_actexp (f01hac)
   * Action of the exponential of a complex matrix on a complex matrix
   */
  nag_matop_complex_gen_matrix_actexp(n, m, a, lda, b, ldb, t, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_complex_gen_matrix_actexp (f01hac)\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, m,
                           b, ldb, "exp(tA) B", NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04cac)\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }

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

10.2 Program Data

```
nag_matop_complex_gen_matrix_actexp (f01hac) Example Program Data

4    2    (-0.5,0.0)                              :Values of n, m and t

(0.5,0.0)    (-0.2,0.0)    (1.0,0.1)    (0.0,0.4)
(0.3,0.0)    ( 0.5,1.2)    (3.1,0.0)    (1.0,0.2)
(0.0,2.0)    ( 0.1,0.0)    (1.2,0.2)    (0.5,0.0)
(1.0,0.3)    ( 0.0,0.2)    (0.0,0.9)    (0.5,0.0) :End of matrix a
```

```
(0.4,0.0)    ( 1.2,0.0)
(1.3,0.0)    (-0.2,0.1)
(0.0,0.3)    ( 2.1,0.0)
(0.4,0.0)    (-0.9,0.0)                          :End of matrix b
```

10.3 Program Results

```
nag_matop_complex_gen_matrix_actexp (f01hac) Example Program Results

 exp(tA) B
                1           2
 1       0.4251     -0.0220
        -0.1061      0.3289

 2       0.7229     -1.7931
        -0.5940      1.4952

 3      -0.1394      1.4781
        -0.1151     -0.4514

 4       0.1054     -1.0059
        -0.0786     -0.7079
```