NAG Library Function Document

nag_matop_complex_herm_matrix_exp (f01fdc)

1 Purpose

nag_matop_complex_herm_matrix_exp (f01fdc) computes the matrix exponential, e^A, of a complex Hermitian n by n matrix A.

2 Specification

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

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

3 Description

e^A is computed using a spectral factorization of A

$$A = QDQ^{\mathrm{H}},$$

where D is the diagonal matrix whose diagonal elements, d_i, are the eigenvalues of A, and Q is a unitary matrix whose columns are the eigenvectors of A. e^A is then given by

$$e^A = Qe^DQ^{\mathrm{H}},$$

where e^D is the diagonal matrix whose ith diagonal element is e^{d_i}. See for example Section 4.5 of Higham (2008).

4 References

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: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangle of the matrix A is stored.

If **uplo** = Nag_Lower, the lower triangle of the matrix A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

3: **n** – Integer *Input*

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

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

4: $\mathbf{a}[dim]$ – Complex *Input/Output*

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

On entry: the n by n Hermitian matrix A.

If **order** = Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag_Lower, the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **fail.code** = NE_NOERROR, the upper or lower triangular part of the n by n matrix exponential, e^A.

5: **pda** – Integer *Input*

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

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

6: **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 $\langle value \rangle$ had an illegal value.

NE_CONVERGENCE

The computation of the spectral factorization failed to converge.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

NE_INT_2

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \mathbf{n}$.

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.

7 Accuracy

For an Hermitian matrix A, the matrix e^A, has the relative condition number

$$\kappa(A) = \|A\|_2,$$

which is the minimal possible for the matrix exponential and so the computed matrix exponential is guaranteed to be close to the exact matrix. See Section 10.2 of Higham (2008) for details and further discussion.

8 Parallelism and Performance

nag_matop_complex_herm_matrix_exp (f01fdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_complex_herm_matrix_exp (f01fdc) 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**, the double allocatable memory required is **n** and the Complex allocatable memory required is approximately $(\mathbf{n} + nb + 1) \times \mathbf{n}$, where nb is the block size required by nag_zheev (f08fnc).

The cost of the algorithm is $O(n^3)$.

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 Hermitian matrix

$$A = \begin{pmatrix} 1 & 2+\ i & 3+2i & 4+3i \\ 2-\ i & 1 & 2+\ i & 3+2i \\ 3-2i & 2-\ i & 1 & 2+\ i \\ 4-3i & 3-2i & 2-\ i & 1 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_complex_herm_matrix_exp (f01fdc) 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 */
  char           *outfile = 0;
  Integer        exit_status = 0;
  Integer        i, j, n, pda;

  /* Arrays */
  char           uplo_c[40];
  Complex        *a = 0;

  /* NAG types */
  Nag_OrderType  order;
  NagError       fail;
  Nag_UploType   uplo;
  Nag_MatrixType matrix;

  INIT_FAIL(fail);

  printf("nag_matop_complex_herm_matrix_exp (f01fdc) Example Program Results");
  printf("\n\n");
  fflush(stdout);

  /* Read matrix dimension and storage from data file*/
#ifdef _WIN32
  scanf_s("%*[^\n]%"NAG_IFMT"%*[^\n] %39s%*[^\n]", &n, uplo_c, _countof(uplo_c));
#else
  scanf("%*[^\n]%"NAG_IFMT"%*[^\n] %39s%*[^\n]", &n, uplo_c);
#endif

  /* nag_enum_name_to_value (x04nac): Converts NAG enum member name to value */
  uplo = (Nag_UploType) nag_enum_name_to_value(uplo_c);

  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*/
  if (uplo == Nag_Upper)
    {
      matrix = Nag_UpperMatrix;
      for (i = 1; i <= n; i++)
        for (j = i; 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
    }
  else
    {
      matrix = Nag_LowerMatrix;
      for (i = 1; i <= n; i++)
        for (j = 1; j <= i; 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_herm_matrix_exp (f01fdc).
   * Complex Hermitian matrix exponential
   */
  nag_matop_complex_herm_matrix_exp(order, uplo, 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 (easy-to-use)
   */
  nag_gen_complx_mat_print(order, matrix, Nag_NonUnitDiag, n, n, a, pda,
                           "Hermitian Exp(A)", outfile, &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_herm_matrix_exp (f01fdc) Example Program Data

   4                                                :Value of N
   Nag_Upper                                        :Value of UPLO

  (1.0, 0.0)  (2.0, 1.0)  (3.0, 2.0)  (4.0, 3.0)
              (1.0, 0.0)  (2.0, 1.0)  (3.0, 2.0)
                          (1.0, 0.0)  (2.0, 1.0)
                                      (1.0, 0.0)  :End of matrix A
```

10.3 Program Results

```
nag_matop_complex_herm_matrix_exp (f01fdc) Example Program Results

 Hermitian Exp(A)
              1             2             3             4
 1    1.1457e+04    8.7983e+03    7.8120e+03    8.3103e+03
      0.0000e+00    2.0776e+03    4.5500e+03    7.8871e+03
```

```
 2                        7.1339e+03    6.8242e+03    7.8120e+03
                          0.0000e+00    2.0776e+03    4.5500e+03

 3                                      7.1339e+03    8.7983e+03
                                        0.0000e+00    2.0776e+03

 4                                                    1.1457e+04
                                                      0.0000e+00
```