NAG Library Function Document

nag_matop_complex_gen_matrix_fun_std (f01fkc)

1 Purpose

nag_matop_complex_gen_matrix_fun_std (f01fkc) computes the matrix exponential, sine, cosine, sinh or cosh, of a complex n by n matrix A using the Schur–Parlett algorithm.

2 Specification

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

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

3 Description

$f(A)$, where f is either the exponential, sine, cosine, sinh or cosh, is computed using the Schur–Parlett algorithm described in Higham (2008) and Davies and Higham (2003).

4 References

Davies P I and Higham N J (2003) A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.* **25(2)** 464–485

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 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: **fun** – Nag_MatFunType *Input*

On entry: indicates which matrix function will be computed.

fun = Nag_Exp
The matrix exponential, e^A, will be computed.

fun = Nag_Sin
The matrix sine, $\sin(A)$, will be computed.

fun = Nag_Cos
The matrix cosine, $\cos(A)$, will be computed.

fun = Nag_Sinh
The hyperbolic matrix sine, $\sinh(A)$, will be computed.

fun = Nag_Cosh
The hyperbolic matrix cosine, $\cosh(A)$, will be computed.

Constraint: **fun** = Nag_Exp, Nag_Sin, Nag_Cos, Nag_Sinh or Nag_Cosh.

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 $\mathbf{pda} \times \mathbf{n}$.

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

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

On entry: the n by n matrix A.

On exit: the n by n matrix, $f(A)$.

5: **pda** – Integer *Input*

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

Allocation of memory failed.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_CONVERGENCE

A Taylor series failed to converge.

NE_INT

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

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 internal error occurred when evaluating the function at a point. Please contact NAG.

An unexpected internal error occurred when ordering the eigenvalues of A. Please contact NAG.

The function was unable to compute the Schur decomposition of A.
Note: this failure should not occur and suggests that the function has been called incorrectly.

There was an error whilst reordering the Schur form of A.
Note: this failure should not occur and suggests that the function has been called incorrectly.

NE_SINGULAR

The linear equations to be solved are nearly singular and the Padé approximant used to compute the exponential may have no correct figures.
Note: this failure should not occur and suggests that the function has been called incorrectly.

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 f at the eigenvalues of A and then constructing $f(A)$ using the Schur vectors. This should give a very accurate result. In general, however, no error bounds are available for the algorithm.

For further discussion of the Schur–Parlett algorithm see Section 9.4 of Higham (2008).

8 Parallelism and Performance

nag_matop_complex_gen_matrix_fun_std (f01fkc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_complex_gen_matrix_fun_std (f01fkc) 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.

In these implementations, this may make calls to the user supplied functions from within an OpenMP parallel region. Thus OpenMP directives within the user functions should be avoided, unless you are using the same OpenMP runtime library (which normally means using the same compiler) as that used to build your NAG Library implementation, as listed in the Installers' Note.

Please 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 $9n^2$.

The cost of the Schur–Parlett algorithm depends on the spectrum of A, but is roughly between $28n^3$ and $n^4/3$ floating-point operations; see Algorithm 9.6 of Higham (2008).

If the matrix exponential is required then it is recommended that nag_matop_complex_gen_matrix_exp (f01fcc) be used. nag_matop_complex_gen_matrix_exp (f01fcc) uses an algorithm which is, in general, more accurate than the Schur–Parlett algorithm used by nag_matop_complex_gen_matrix_fun_std (f01fkc).

If estimates of the condition number of the matrix function are required then nag_matop_complex_gen_matrix_cond_std (f01kac) should be used.

nag_matop_real_gen_matrix_fun_std (f01ekc) can be used to find the matrix exponential, sin, cos, sinh or cosh of a real matrix A.

10 Example

This example finds the matrix sinh of the matrix

$$A = \begin{pmatrix} 1.0+1.0i & 0.0+0.0i & 1.0+3.0i & 0.0+0.0i \\ 0.0+0.0i & 2.0+0.0i & 0.0+0.0i & 1.0+2.0i \\ 3.0+1.0i & 0.0+4.0i & 1.0+1.0i & 0.0+0.0i \\ 1.0+1.0i & 0.0+2.0i & 0.0+0.0i & 1.0+0.0i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_fun_std (f01fkc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#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;
  char           nag_enum_arg[10];

  /* Nag Types */
  Nag_OrderType  order;
  Nag_MatFunType fun;
  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_fun_std (f01fkc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
  scanf("%*[^\n]");

  /* Read in the problem size and the required function */
  scanf("%ld%9s%*[^\n]", &n, nag_enum_arg);
  pda = n;

  /* nag_enum_name_to_value (x04nac)
   * Converts Nag enum member name to value
   */
  fun = (Nag_MatFunType) nag_enum_name_to_value(nag_enum_arg);

  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++)
    for (j = 1; j <= n; j++) scanf(" ( %lf , %lf ) ", &A(i,j).re, &A(i,j).im);
  scanf("%*[^\n]");

  /* Find the matrix function using
   * nag_matop_complex_gen_matrix_fun_std (f01fkc)
   * Complex matrix function
   */
  nag_matop_complex_gen_matrix_fun_std(order, fun, n, a, pda, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_matop_complex_gen_matrix_fun_std (f01fkc)\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, "f(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_fun_std (f01fkc) Example Program Data
 4   Nag_Sinh                                  :Values of n and fun
(1.0, 1.0) ( 0.0, 0.0) ( 1.0, 3.0) (0.0, 0.0)
(0.0, 0.0) ( 2.0, 0.0) ( 0.0, 0.0) (1.0, 2.0)
(3.0, 1.0) ( 0.0, 4.0) ( 1.0, 1.0) (0.0, 0.0)
(1.0, 1.0) ( 0.0, 2.0) ( 0.0, 0.0) (1.0, 0.0) :End of matrix a
```

10.3 Program Results

```
nag_matop_complex_gen_matrix_fun_std (f01fkc) Example Program Results

 f(A)
             1          2          3          4
 1     -4.3015    -1.4918    -4.4242     1.4438
       -1.8117    -8.7793    -1.3925    -6.5287

 2     -1.7976     1.4211    -1.2712     1.2118
       -0.2935    -0.1993    -1.9931     2.8506

 3     -4.4968    -5.7934    -4.3015    -3.0082
       -0.1964    -4.7166    -1.8117    -4.1821

 4     -2.1506    -0.6103    -1.5163     0.0385
       -0.3911    -1.4408    -1.9317    -0.2847
```