NAG Library Function Document

nag_matop_complex_tri_matrix_sqrt (f01fpc)

1 Purpose

nag_matop_complex_tri_matrix_sqrt (f01fpc) computes the principal matrix square root, $A^{1/2}$, of a complex upper triangular n by n matrix A.

2 Specification

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

3 Description

A square root of a matrix A is a solution X to the equation $X^2 = A$. A nonsingular matrix has multiple square roots. For a matrix with no eigenvalues on the closed negative real line, the principal square root, denoted by $A^{1/2}$, is the unique square root whose eigenvalues lie in the open right half-plane.

nag_matop_complex_tri_matrix_sqrt (f01fpc) computes $A^{1/2}$, where A is an upper triangular matrix. $A^{1/2}$ is also upper triangular.

The algorithm used by nag_matop_complex_tri_matrix_sqrt (f01fpc) is described in BjÎrck and Hammarling (1983). In addition a blocking scheme described in Deadman *et al.* (2013) is used.

4 References

BjÎrck Đ and Hammarling S (1983) A Schur method for the square root of a matrix *Linear Algebra Appl.* **52/53** 127–140

Deadman E, Higham N J and Ralha R (2013) Blocked Schur Algorithms for Computing the Matrix Square Root *Applied Parallel and Scientific Computing: 11th International Conference, (PARA 2012, Helsinki, Finland)* P. Manninen and P. Úster, Eds *Lecture Notes in Computer Science* **7782** 171–181 Springer–Verlag

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: **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** $+ i - 1$].

On entry: the n by n upper triangular matrix A.

On exit: contains, if **fail.code** = NE_NOERROR, the n by n principal matrix square root, $A^{1/2}$. Alternatively, if **fail.code** = NE_EIGENVALUES, contains an n by n non-principal square root of A.

3: **pda** – Integer *Input*

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

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

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

NE_EIGENVALUES

A has negative or semisimple, vanishing eigenvalues. The principal square root is not defined in this case; a non-principal square root is returned.

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pda** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
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 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.

NE_SINGULAR

A has a defective vanishing eigenvalue. The square root cannot be found in this case.

7 Accuracy

The computed square root $\hat{X}$ satisfies $\hat{X}^2 = A + \Delta A$, where $|\Delta A| \approx O(\epsilon)n\left|\hat{X}\right|^2$, where ϵ is ***machine precision***. The order of the change in A is to be interpreted elementwise.

8 Parallelism and Performance

nag_matop_complex_tri_matrix_sqrt (f01fpc) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_matop_complex_tri_matrix_sqrt (f01fpc) 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 cost of the algorithm is $n^3/3$ complex floating-point operations; see Algorithm 6.3 in Higham (2008). $O\left(2 \times n^2\right)$ of complex allocatable memory is required by the function.

If A is a full matrix, then nag_matop_complex_gen_matrix_sqrt (f01fnc) should be used to compute the principal square root.

If condition number and residual bound estimates are required, then nag_matop_complex_gen_matrix_cond_sqrt (f01kdc) should be used. For further discussion of the condition of the matrix square root see Section 6.1 of Higham (2008).

10 Example

This example finds the principal matrix square root of the matrix

$$A = \begin{pmatrix} 2i & 14+2i & 12+3i & 6+4i \\ 0 & -5+12i & 6+18i & 9+16i \\ 0 & 0 & 3-4i & 16-4i \\ 0 & 0 & 0 & 4 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_complex_tri_matrix_sqrt (f01fpc) 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 <nagx04.h>

#define A(I,J) a[J*pda + I]

int main(void)
{

  /* Scalars */
  Integer exit_status = 0;
  Integer i, j, n, pda;
  /* Arrays */
  Complex *a = 0;
  /* Nag Types */
  Nag_OrderType order = Nag_ColMajor;
  NagError fail;

  INIT_FAIL(fail);
```

```
  /* Output preamble */
  printf("nag_matop_complex_tri_matrix_sqrt (f01fpc) ");
  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 = 0; i < n; i++) {
    for (j = 0; j < i; j++) {
      A(i, j).re = 0.0;
      A(i, j).im = 0.0;
    }
    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
    }
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Find matrix square root using
   * nag_matop_complex_tri_matrix_sqrt (f01fpc)
   * Complex upper triangular matrix square root
   */
  nag_matop_complex_tri_matrix_sqrt(n, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_complex_tri_matrix_sqrt (f01fpc)\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print matrix sqrt(A) 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, pda, "sqrt(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_tri_matrix_sqrt (f01fpc) Example Program Data

4                                                       :Value of n

( 0.0,  2.0) ( 14.0,  2.0) ( 12.0,  3.0) ( 6.0, 4.0)
(-5.0, 12.0) (  6.0, 18.0) (  9.0, 16.0)
( 3.0, -4.0) ( 16.0, -4.0)
( 4.0,  0.0)                                            :End of matrix a
```

10.3 Program Results

```
nag_matop_complex_tri_matrix_sqrt (f01fpc) Example Program Results

 sqrt(A)
              1           2           3           4
 1       1.0000      2.0000     -0.0000      1.0000
         1.0000     -2.0000      1.0000     -1.0000

 2       0.0000      2.0000      3.0000      0.0000
         0.0000      3.0000      3.0000      1.0000

 3       0.0000      0.0000      2.0000      4.0000
         0.0000      0.0000     -1.0000      0.0000

 4       0.0000      0.0000      0.0000      2.0000
         0.0000      0.0000      0.0000      0.0000
```