NAG Library Function Document

nag_matop_real_tri_matrix_sqrt (f01epc)

1 Purpose

nag_matop_real_tri_matrix_sqrt (f01epc) computes the principal matrix square root, $A^{1/2}$, of a real upper quasi-triangular n by n matrix A.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_real_tri_matrix_sqrt (Integer n, double 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_real_tri_matrix_sqrt (f01epc) computes $A^{1/2}$, where A is an upper quasi-triangular matrix, with 1×1 and 2×2 blocks on the diagonal. Such matrices arise from the Schur factorization of a real general matrix, as computed by nag_dhseqr (f08pec), for example. nag_matop_real_tri_matrix_sqrt (f01epc) does not require A to be in the canonical Schur form described in nag_dhseqr (f08pec), it merely requires A to be upper quasi-triangular. $A^{1/2}$ then has the same block triangular structure as A.

The algorithm used by nag_matop_real_tri_matrix_sqrt (f01epc) is described in Higham (1987). In addition a blocking scheme described in Deadman *et al.* (2013) is used.

4 References

Björck Å 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. Öster, Eds *Lecture Notes in Computer Science* **7782** 171–181 Springer–Verlag

Higham N J (1987) Computing real square roots of a real matrix *Linear Algebra Appl.* **88/89** 405–430

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

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

On exit: the n by n principal matrix square root $A^{1/2}$.

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 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

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

NE_EIGENVALUES

A has negative or vanishing eigenvalues. The principal square root is not defined in this case. nag_matop_real_gen_matrix_sqrt (f01enc) or nag_matop_complex_gen_matrix_sqrt (f01fnc) may be able to provide further information.

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.

7 Accuracy

The computed square root $\hat{X}$ satisfies $\hat{X}^2 = A + \Delta A$, where $\|\Delta A\|_F \approx O(\epsilon) n \|\hat{X}\|_F^2$, where ϵ is ***machine precision***.

8 Parallelism and Performance

nag_matop_real_tri_matrix_sqrt (f01epc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_real_tri_matrix_sqrt (f01epc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The cost of the algorithm is $n^3/3$ floating-point operations; see Algorithm 6.7 of Higham (2008). $O(n)$ of integer allocatable memory is required by the function.

If A is a full matrix, then nag_matop_real_gen_matrix_sqrt (f01enc) should be used to compute the square root. If A has negative real eigenvalues then nag_matop_complex_gen_matrix_sqrt (f01fnc) can be used to return a complex, non-principal square root.

If condition number and residual bound estimates are required, then nag_matop_real_gen_matrix_cond_sqrt (f01jdc) 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

$$A = \begin{pmatrix} 6 & 4 & -5 & 15 \\ 8 & 6 & -3 & 10 \\ 0 & 0 & 3 & -4 \\ 0 & 0 & 4 & 3 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_real_tri_matrix_sqrt (f01epc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#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 */
  double         *a = 0;
  /* Nag Types */
  Nag_OrderType  order = Nag_ColMajor;
  NagError       fail;

  INIT_FAIL(fail);

  /* Output preamble */
  printf("nag_matop_real_tri_matrix_sqrt (f01epc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

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

  /* Read in the problem size */
  scanf("%ld%*[^\n]", &n);

  pda = n;
  if (!(a = NAG_ALLOC(pda*n, double))) {
    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 < n; j++) {
        scanf("%lf", &A(i, j));
     }
  }
  scanf("%*[^\n] ");

  /* Find matrix square root using
   * nag_matop_real_tri_matrix_sqrt (f01epc)
   * Real upper quasi-triangular matrix square root
   */
  nag_matop_real_tri_matrix_sqrt (n, a, pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_real_tri_matrix_sqrt (f01epc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print matrix sqrt(A) using
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_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_real_mat_print (x04cac)\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

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

10.2 Program Data

```
nag_matop_real_tri_matrix_sqrt (f01epc) Example Program Data

4                               :Value of n

6.0     4.0    -5.0     15.0
8.0     6.0    -3.0     10.0
0.0     0.0     3.0     -4.0
0.0     0.0     4.0      3.0  :End of matrix a
```

10.3 Program Results

```
nag_matop_real_tri_matrix_sqrt (f01epc) Example Program Results

 sqrt(A)
              1            2            3            4
 1    2.0000e+00   1.0000e+00  -2.0000e+00   3.0000e+00
 2    2.0000e+00   2.0000e+00   3.1721e-16   1.0000e+00
 3    0.0000e+00   0.0000e+00   2.0000e+00  -1.0000e+00
 4    0.0000e+00   0.0000e+00   1.0000e+00   2.0000e+00
```