NAG Library Function Document

nag_matop_complex_gen_matrix_pow (f01fqc)

1 Purpose

nag_matop_complex_gen_matrix_pow (f01fqc) computes an abitrary real power A^p of a complex n by n matrix A.

2 Specification

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

3 Description

For a matrix A with no eigenvalues on the closed negative real line, A^p ($p \in \mathbb{R}$) can be defined as

$$A^p = \exp(p\log(A))$$

where $\log(A)$ is the principal logarithm of A (the unique logarithm whose spectrum lies in the strip $\{z : -\pi < \mathrm{Im}(z) < \pi\}$).

A^p is computed using the Schur–Padé algorithm described in Higham and Lin (2011) and Higham and Lin (2013).

The real number p is expressed as $p = q + r$ where $q \in (-1, 1)$ and $r \in \mathbb{Z}$. Then $A^p = A^q A^r$. The integer power A^r is found using a combination of binary powering and, if necessary, matrix inversion. The fractional power A^q is computed using a Schur decomposition and a Padé approximant.

4 References

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

Higham N J and Lin L (2011) A Schur–Padé algorithm for fractional powers of a matrix *SIAM J. Matrix Anal. Appl.* **32(3)** 1056–1078

Higham N J and Lin L (2013) An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives *MIMS Eprint 2013.1* Manchester Institute for Mathematical Sciences, School of Mathematics, University of Manchester http://eprints.ma.man.ac.uk/

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

On entry: the n by n matrix A.

On exit: if **fail.code** = NE_NOERROR, the n by n matrix pth power, A^p. Alternatively, if **fail.code** = NE_NEGATIVE_EIGVAL, contains an n by n non-principal power of A.

3: **pda** – Integer *Input*

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

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

4: **p** – double *Input*

On entry: the required power of A.

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

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.

NE_NEGATIVE_EIGVAL

A has eigenvalues on the negative real line. The principal pth power is not defined so a non-principal power is returned.

NE_SINGULAR

A is singular so the pth power cannot be computed.

NW_SOME_PRECISION_LOSS

A^p has been computed using an IEEE double precision Padé approximant, although the arithmetic precision is higher than IEEE double precision.

7 Accuracy

For positive integer p, the algorithm reduces to a sequence of matrix multiplications. For negative integer p, the algorithm consists of a combination of matrix inversion and matrix multiplications.

For a normal matrix A (for which $A^{\mathrm{H}}A = AA^{\mathrm{H}}$) and non-integer p, the Schur decomposition is diagonal and the algorithm reduces to evaluating powers of the eigenvalues of A and then constructing A^p using the Schur vectors. This should give a very accurate result. In general however, no error bounds are available for the algorithm.

8 Parallelism and Performance

nag_matop_complex_gen_matrix_pow (f01fqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_matop_complex_gen_matrix_pow (f01fqc) 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 $O(n^3)$. The exact cost depends on the matrix A but if $p \in (-1, 1)$ then the cost is independent of p. $O(4 \times n^2)$ complex allocatable memory is required by the function.

If estimates of the condition number of A^p are required then nag_matop_complex_gen_matrix_cond_pow (f01kec) should be used.

10 Example

This example finds A^p where $p = 0.2$ and

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

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_pow (f01fqc) 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;
  double         p;
  /* Arrays */
  Complex        *a = 0;
  /* Nag Types */
  Nag_OrderType  order = Nag_ColMajor;
  NagError       fail;

  INIT_FAIL(fail);

  /* Output preamble */
  printf("nag_matop_complex_gen_matrix_pow (f01fqc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
```

```
  scanf("%*[^\n] ");

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

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

  /* Find the matrix pth power using
   * nag_matop_complex_gen_matrix_pow (f01fqc)
   * General power of a complex matrix
   */
  nag_matop_complex_gen_matrix_pow (n, a, pda, p, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_complex_gen_matrix_pow (f01fqc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print matrix A^p 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, "A^p", 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_pow (f01fqc) Example Program Data

4     0.2                                       : Values of n and p

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

10.3 Program Results

```
nag_matop_complex_gen_matrix_pow (f01fqc) Example Program Results

 A^p
             1           2           3           4
 1      1.1766      0.1375      0.2742     -0.1435
       -0.0758      0.2241     -0.2223      0.0816
```

```
 2      0.2074      1.1118     -0.1343      0.1794
       -0.0998     -0.0039     -0.0404      0.3590

 3     -0.0859      0.5224      1.0616      0.2308
       -0.0824     -0.0530      0.3921      0.1856

 4      0.3313     -0.1507      0.2178      1.1710
        0.1303      0.0982     -0.0061     -0.2136
```