NAG Library Function Document

nag_matop_complex_gen_matrix_fun_num (f01flc)

1 Purpose

nag_matop_complex_gen_matrix_fun_num (f01flc) computes the matrix function, $f(A)$, of a complex n by n matrix A. Numerical differentiation is used to evaluate the derivatives of f when they are required.

2 Specification

```
#include <nag.h>
#include <nagf01.h>
void nag_matop_complex_gen_matrix_fun_num (Integer n, Complex a[],
     Integer pda,
     void (*f)(Integer *iflag, Integer nz, const Complex z[], Complex fz[],
          Nag_Comm *comm),
     Nag_Comm *comm, Integer *iflag, NagError *fail)
```

3 Description

$f(A)$ is computed using the Schur–Parlett algorithm described in Higham (2008) and Davies and Higham (2003). The coefficients of the Taylor series used in the algorithm are evaluated using the numerical differentiation algorithm of Lyness and Moler (1967).

The scalar function f is supplied via function **f** which evaluates $f(z_i)$ at a number of points z_i.

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

Lyness J N and Moler C B (1967) Numerical differentiation of analytic functions *SIAM J. Numer. Anal.* **4(2)** 202–210

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

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 matrix A.

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

3: **pda** – Integer *Input*

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

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

4: **f** – function, supplied by the user *External Function*

The function **f** evaluates $f(z_i)$ at a number of points z_i.

The specification of **f** is:

```
void f (Integer *iflag, Integer nz, const Complex z[], Complex fz[],
     Nag_Comm *comm)
```

1: **iflag** – Integer * *Input/Output*

On entry: **iflag** will be zero.

On exit: **iflag** should either be unchanged from its entry value of zero, or may be set nonzero to indicate that there is a problem in evaluating the function $f(z_i)$; for instance $f(z_i)$ may not be defined. If **iflag** is returned as nonzero then nag_matop_complex_gen_matrix_fun_num (f01flc) will terminate the computation, with **fail.code** = NE_USER_STOP.

2: **nz** – Integer *Input*

On entry: n_z, the number of function values required.

3: **z**[**nz**] – const Complex *Input*

On entry: the n_z points $z_1, z_2, \ldots, z_{n_z}$ at which the function f is to be evaluated.

4: **fz**[**nz**] – Complex *Output*

On exit: the n_z function values. **fz**$[i-1]$ should return the value $f(z_i)$, for $i = 1, 2, \ldots, n_z$.

5: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_matop_complex_gen_matrix_fun_num (f01flc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_matop_complex_gen_matrix_fun_num (f01flc) (see Section 3.2.1.1 in the Essential Introduction).

5: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

6: **iflag** – Integer * *Output*

On exit: **iflag** = 0, unless **iflag** has been set nonzero inside **f**, in which case **iflag** will be the value set and **fail** will be set to **fail.code** = NE_USER_STOP.

7: **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 ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

A Taylor series failed to converge after 40 terms. Further Taylor series coefficients can no longer reliably be obtained by numerical differentiation.

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 error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

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_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.

NE_USER_STOP

iflag has been set nonzero by the user.

7 Accuracy

For a normal matrix A (for which $A^{\mathrm{H}}A = AA^{\mathrm{H}}$) 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. See Section 9.4 of Higham (2008) for further discussion of the Schur–Parlett algorithm, and Lyness and Moler (1967) for a discussion of numerical differentiation.

8 Parallelism and Performance

nag_matop_complex_gen_matrix_fun_num (f01flc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library. In these implementations, this function may make calls to the user-supplied functions from within an OpenMP parallel region. Thus OpenMP pragmas within the user functions can only be used if you are compiling the user-supplied function and linking

the executable in accordance with the instructions in the Users' Note for your implementation. You must also ensure that you use the NAG communication argument **comm** in a thread safe manner, which is best achieved by only using it to supply read-only data to the user functions.

nag_matop_complex_gen_matrix_fun_num (f01flc) 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, and up to $6n^2$ of Complex allocatable memory is required.

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. There is an additional cost in numerically differentiating f, in order to obtain the Taylor series coefficients. If the derivatives of f are known analytically, then nag_matop_complex_gen_matrix_fun_usd (f01fmc) can be used to evaluate $f(A)$ more accurately. If A is complex Hermitian then it is recommended that nag_matop_complex_herm_matrix_fun (f01ffc) be used as it is more efficient and, in general, more accurate than nag_matop_complex_gen_matrix_fun_num (f01flc).

Note that f must be analytic in the region of the complex plane containing the spectrum of A.

For further information on matrix functions, see Higham (2008).

If estimates of the condition number of the matrix function are required then nag_matop_complex_gen_matrix_cond_num (f01kbc) should be used.

nag_matop_real_gen_matrix_fun_num (f01elc) can be used to find the matrix function $f(A)$ for a real matrix A.

10 Example

This example finds $\sin 2A$ where

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

10.1 Program Text

```
/* nag_matop_complex_gen_matrix_fun_num (f01flc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <nagf01.h>
#include <nagx04.h>
#include <math.h>

#ifdef __cplusplus
extern "C" {
#endif
  static void NAG_CALL f(Integer *iflag, Integer nz, const Complex z[],
                         Complex fz[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif
```

```
int main(void)
{
  /* Scalars */
  Integer        exit_status = 0;
  Integer        i, iflag, j, n, pda;

  /* Arrays  */
  static double ruser[1] = {-1.0};
  Complex        *a = 0;

  /* Nag Types */
  Nag_Comm       comm;
  Nag_OrderType  order;
  NagError       fail;

  INIT_FAIL(fail);

  #define A(I, J)  a[(J-1)*pda + I-1]

  order = Nag_ColMajor;

  /* Output preamble .. */
  printf("nag_matop_complex_gen_matrix_fun_num (f01flc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* 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 = 1; i <= n; i++)
    for (j = 1; 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 the matrix function using
   * nag_matop_complex_gen_matrix_fun_num (f01flc)
   * Function of a complex matrix
   */
  nag_matop_complex_gen_matrix_fun_num(n, a, pda, f, &comm, &iflag, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_complex_gen_matrix_fun_num (f01flc)\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,
                           pda, "f(A) = sin(2A)", 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;
}

static void NAG_CALL f(Integer *iflag, Integer nz, const Complex z[],
                       Complex fz[], Nag_Comm *comm)
{
  /* Scalars */
  Integer j;
#pragma omp master
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback f, first invocation.)\n");
      fflush(stdout);
      comm->user[0] = 0.0;
    }
  for (j = 0; j < nz; j++) {
    /* Implementation of sin2z for complex z */
    fz[j].re = sin(2.0*z[j].re)*cosh(2.0*z[j].im);
    fz[j].im = cos(2.0*z[j].re)*sinh(2.0*z[j].im);
  }
  /* Set iflag nonzero to terminate execution for any reason. */
  *iflag = 0;
}
```

10.2 Program Data

```
nag_matop_complex_gen_matrix_fun_num (f01flc) Example Program Data
  4                                                  :Value of n
( 1.0, 0.0)  (0.0, 1.0)  ( 1.0, 0.0)  (0.0, 1.0)
(-1.0, 0.0)  (0.0, 0.0)  ( 2.0, 1.0)  (0.0, 0.0)
( 0.0, 0.0)  (2.0, 1.0)  ( 0.0, 2.0)  (0.0, 1.0)
( 1.0, 0.0)  (1.0, 1.0)  (-1.0, 0.0)  (2.0, 1.0)  :End of matrix a
```

10.3 Program Results

```
nag_matop_complex_gen_matrix_fun_num (f01flc) Example Program Results

(User-supplied callback f, first invocation.)
 f(A) = sin(2A)
             1           2           3           4
 1      1.1960    -21.0733    -15.4159    -12.4279
       -3.2270     -9.6441    -14.1977    -11.9638

 2      3.2957    -14.6084     -6.7764     -5.1338
       -3.6334    -21.4846    -24.1726    -17.0926

 3      5.0928    -14.6839     -0.9231     -2.0715
```

```
        -3.7806    -34.5063    -35.4729    -26.3460

 4      -1.8349     -8.2484     -6.0093     -7.1318
         0.0808     -0.4014     -1.6831     -1.9396
```