NAG Library Function Document

nag_matop_real_gen_matrix_fun_usd (f01emc)

1 Purpose

nag_matop_real_gen_matrix_fun_usd (f01emc) computes the matrix function, $f(A)$, of a real n by n matrix A, using analytical derivatives of f you have supplied.

2 Specification

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

3 Description

$f(A)$ is computed using the Schur–Parlett algorithm described in Higham (2008) and Davies and Higham (2003).

The scalar function f, and the derivatives of f, are returned by the function **f** which, given an integer m, should evaluate $f^{(m)}(z_i)$ at a number of (generally complex) points z_i, for $i = 1, 2, \ldots, n_z$. For any z on the real line, $f(z)$ must also be real. nag_matop_real_gen_matrix_fun_usd (f01emc) is therefore appropriate for functions that can be evaluated on the complex plane and whose derivatives, of arbitrary order, can also be evaluated on the complex plane.

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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: **n** ≥ 0.

3: **a**[dim] – double *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** $+ i - 1]$ when **order** = Nag_ColMajor;
a$[(i-1) \times$ **pda** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by n matrix A.

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

4: **pda** – Integer *Input*

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

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

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

Given an integer m, the function **f** evaluates $f^{(m)}(z_i)$ at a number of points z_i.

The specification of **f** is:

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

1: **m** – Integer *Input*

On entry: the order, m, of the derivative required.

If **m** $= 0$, $f(z_i)$ should be returned. For **m** > 0, $f^{(m)}(z_i)$ should be returned.

2: **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 $f(z)$; for instance $f(z_i)$ may not be defined for a particular z_i. If **iflag** is returned as nonzero then nag_matop_real_gen_matrix_fun_usd (f01emc) will terminate the computation, with **fail**.**code** = NE_USER_STOP.

3: **nz** – Integer *Input*

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

4: **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.

5: **fz**[**nz**] – Complex *Output*

On exit: the n_z function or derivative values. **fz**$[i-1]$ should return the value $f^{(m)}(z_i)$, for $i = 1, 2, \ldots, n_z$. If z_i lies on the real line, then so must $f^{(m)}(z_i)$.

6: **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_real_gen_matrix_fu n_usd (f01emc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_matop_real_gen_matrix_fu

n_usd (f01emc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

6: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7: **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.

8: **imnorm** – double * *Output*

On exit: if A has complex eigenvalues, nag_matop_real_gen_matrix_fun_usd (f01emc) will use complex arithmetic to compute $f(A)$. The imaginary part is discarded at the end of the computation, because it will theoretically vanish. **imnorm** contains the 1-norm of the imaginary part, which should be used to check that the function has given a reliable answer.

If A has real eigenvalues, nag_matop_real_gen_matrix_fun_usd (f01emc) uses real arithmetic and **imnorm** $= 0$.

9: **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 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_CONVERGENCE

A Taylor series failed to converge.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

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

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.

The routine was unable to compute the Schur decomposition 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 How to Use the NAG Library and its Documentation for further information.

NE_USER_STOP

iflag has been set nonzero by the user.

7 Accuracy

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

8 Parallelism and Performance

nag_matop_real_gen_matrix_fun_usd (f01emc) 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_real_gen_matrix_fun_usd (f01emc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

If A has real eigenvalues then up to $6n^2$ of double allocatable memory may be required. If A has complex eigenvalues then up to $6n^2$ of Complex allocatable memory may be 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 evaluating f and its derivatives. If the derivatives of f are not known analytically, then nag_matop_real_gen_matrix_fun_num (f01elc) can be used to evaluate $f(A)$ using numerical differentiation. If A is real symmetric then it is recommended that nag_matop_real_symm_matrix_fun (f01efc) be used as it is more efficient and, in general, more accurate than nag_matop_real_gen_matrix_fun_usd (f01emc).

For any z on the real line, $f(z)$ must be real. f must also be complex analytic on the spectrum of A. These conditions ensure that $f(A)$ is real for real 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_real_gen_matrix_cond_usd (f01jcc) should be used.

nag_matop_complex_gen_matrix_fun_usd (f01fmc) can be used to find the matrix function $f(A)$ for a complex matrix A.

10 Example

This example finds the e^{2A} where

$$A = \begin{pmatrix} 1 & 0 & -2 & 1 \\ -1 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & -1 & 2 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_matop_real_gen_matrix_fun_usd (f01emc) 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 <nagx02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL f(Integer m, 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;
  double imnorm;

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

  /* Nag Types */
  Nag_Comm comm;
  Nag_OrderType order;
  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_real_gen_matrix_fun_usd (f01emc) ");
  printf("Example Program Results\n\n");

  /* For communication with user-supplied functions: */
```

```
  comm.user = ruser;

  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), double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in the matrix a from data file */
  for (i = 1; i <= n; i++)
#ifdef _WIN32
    for (j = 1; j <= n; j++)
      scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= n; j++)
      scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Find the matrix function using
   * nag_matop_real_gen_matrix_fun_usd (f01emc)
   * Function of a real matrix
   */
  nag_matop_real_gen_matrix_fun_usd(order, n, a, pda, f, &comm, &iflag,
                                    &imnorm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_matop_real_gen_matrix_fun_usd (f01emc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  if (fabs(imnorm) > pow(nag_machine_precision, 0.8)) {
    printf("\nWARNING: the error estimate returned in imnorm is larger than"
           " expected;\n");
    printf("         the matrix solution printed below is suspect:\n");
    printf("         imnorm = %13.4e.\n\n", imnorm);
  }

  /* Print solution using
   * nag_gen_real_mat_print (x04dac)
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
                         pda, "f(A)", NULL, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_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 m, 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++) {
    /* The m^th derivative of exp(2z) for complex z */
    fz[j].re = pow(2.0, m) * exp(2.0 * z[j].re) * cos(2.0 * z[j].im);
    fz[j].im = pow(2.0, m) * exp(2.0 * z[j].re) * sin(2.0 * z[j].im);
  }
  /* Set iflag nonzero to terminate execution for any reason. */
  *iflag = 0;
}
```

10.2 Program Data

```
nag_matop_real_gen_matrix_fun_usd (f01emc) Example Program Data
 4                              :Value of n
 1.0     0.0    -2.0     1.0
-1.0     2.0     0.0     1.0
 2.0     0.0     1.0     0.0
 1.0     0.0    -1.0     2.0  :End of matrix a
```

10.3 Program Results

```
nag_matop_real_gen_matrix_fun_usd (f01emc) Example Program Results

(User-supplied callback f, first invocation.)
 f(A)
             1          2          3          4
 1    -12.1880     0.0000    -3.4747     8.3697
 2    -13.7274    54.5982   -23.9801    82.8593
 3     -9.7900     0.0000   -25.4527    26.5294
 4    -18.1597     0.0000   -34.8991    49.2404
```