NAG Library Routine Document

F01FMF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

F01FMF computes the matrix function, $f(A)$, of a complex n by n matrix A, using analytical derivatives of f you have supplied.

2 Specification

```
SUBROUTINE F01FMF (N, A, LDA, F, IUSER, RUSER, IFLAG, IFAIL)
INTEGER                 N, LDA, IUSER(*), IFLAG, IFAIL
REAL (KIND=nag_wp)      RUSER(*)
COMPLEX (KIND=nag_wp)   A(LDA,*)
EXTERNAL                F
```

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 subroutine F which, given an integer m, should evaluate $f^{(m)}(z_i)$ at a number of points z_i, for $i = 1, 2, \ldots, n_z$, on the complex plane. F01FMF 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 Parameters

1: N – INTEGER *Input*

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

Constraint: N ≥ 0.

2: A(LDA, $*$) – COMPLEX (KIND=nag_wp) array *Input/Output*

Note: the second dimension of the array A must be at least N.

On entry: the n by n matrix A.

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

3: LDA – INTEGER *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which F01FMF is called.

Constraint: LDA $\geq$ N.

4: F – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of F is:

```
SUBROUTINE F (M, IFLAG, NZ, Z, FZ, IUSER, RUSER)
INTEGER                 M, IFLAG, NZ, IUSER(*)
REAL (KIND=nag_wp)      RUSER(*)
COMPLEX (KIND=nag_wp) Z(NZ), FZ(NZ)
```

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 F01FMF will terminate the computation, with IFAIL $= 2$.

3: NZ – INTEGER *Input*

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

4: Z(NZ) – COMPLEX (KIND=nag_wp) array *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 (KIND=nag_wp) array *Output*

On exit: the n_z function or derivative values. FZ(i) should return the value $f^{(m)}(z_i)$, for $i = 1, 2, \ldots, n_z$.

6: IUSER($*$) – INTEGER array *User Workspace*

7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

F is called with the parameters IUSER and RUSER as supplied to F01FMF. You are free to use the arrays IUSER and RUSER to supply information to F as an alternative to using COMMON global variables.

F must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which F01FMF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: IUSER($*$) – INTEGER array *User Workspace*

6: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by F01FMF, but are passed directly to F and may be used to pass information to this routine as an alternative to using COMMON global variables.

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 IFAIL will be set to IFAIL $= 2$.

8: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

A Taylor series failed to converge.

IFAIL $= 2$

IFLAG has been set nonzero by the user.

IFAIL $= 3$

There was an error whilst reordering the Schur form of A.
Note: this failure should not occur and suggests that the routine has been called incorrectly.

IFAIL $= 4$

The routine was unable to compute the Schur decomposition of A.
Note: this failure should not occur and suggests that the routine has been called incorrectly.

IFAIL $= 5$

An unexpected internal error occurred. Please contact NAG.

IFAIL $= -1$

Input argument number $\langle value \rangle$ is invalid.

IFAIL $= -3$

On entry, parameter LDA is invalid.
Constraint: LDA $\geq$ N.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

For a normal matrix A (for which $A^{\mathrm{H}}A = AA^{\mathrm{H}}$), 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

F01FMF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library. In these implementations, this routine may make calls to the user-supplied functions from within an OpenMP parallel region. Thus OpenMP directives 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. The user workspace arrays IUSER and RUSER are classified as OpenMP shared memory and use of IUSER and RUSER has to take account of this in order to preserve thread safety whenever information is written back to either of these arrays. If at all possible, it is recommended that these arrays are only used to supply read-only data to the user functions when a multithreaded implementation is being used.

F01FMF 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 routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

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 evaluating f and its derivatives. If the derivatives of f are not known analytically, then F01FLF can be used to evaluate $f(A)$ using numerical differentiation. If A is complex Hermitian then it is recommended that F01FFF be used as it is more efficient and, in general, more accurate than F01FMF.

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 F01KCF should be used.

F01EMF can be used to find the matrix function $f(A)$ for a real matrix A.

10 Example

This example finds the e^{3A} where

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

10.1 Program Text

```
!   F01FMF Example Program Text

!   Mark 25 Release. NAG Copyright 2014.

    Module f01fmfe_mod

!     F01FMF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                                :: fexp3
    Contains
      Subroutine fexp3(m,iflag,nz,z,fz,iuser,ruser)

!       .. Parameters ..
        Complex (Kind=nag_wp), Parameter     ::                                 &
                                            three = (3.0E0_nag_wp,0.0E0_nag_wp)
!       .. Scalar Arguments ..
        Integer, Intent (Inout)              :: iflag
        Integer, Intent (In)                 :: m, nz
!       .. Array Arguments ..
        Complex (Kind=nag_wp), Intent (Out)  :: fz(nz)
        Complex (Kind=nag_wp), Intent (In)   :: z(nz)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: exp
!       .. Executable Statements ..
        Continue
        fz(1:nz) = (three**m)*exp(three*z(1:nz))
!       Set iflag nonzero to terminate execution for any reason.
        iflag = 0
        Return
      End Subroutine fexp3
    End Module f01fmfe_mod
    Program f01fmfe

!     F01FMF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: f01fmf, nag_wp, x04daf
      Use f01fmfe_mod, Only: fexp3
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                              :: i, ifail, iflag, lda, n
!     .. Local Arrays ..
      Complex (Kind=nag_wp), Allocatable   :: a(:,:)
      Real (Kind=nag_wp)                   :: ruser(1)
      Integer                              :: iuser(1)
!     .. Executable Statements ..
      Write (nout,*) 'F01FMF Example Program Results'
      Write (nout,*)
      Flush (nout)

!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) n

      lda = n
      Allocate (a(lda,n))
```

```
!     Read A from data file
      Read (nin,*)(a(i,1:n),i=1,n)

!     Find f( A )
      ifail = 0
      Call f01fmf(n,a,lda,fexp3,iuser,ruser,iflag,ifail)

!     Print solution
      ifail = 0
      Call x04daf('G','N',n,n,a,lda,'F(A) = EXP(3A)',ifail)

    End Program f01fmfe
```

10.2 Program Data

```
F01FMF Example Program Data

  4                                                    :Value of N

( 1.0, 0.0)  (0.0, 0.0)  ( 1.0, 0.0)  (0.0, 2.0)
( 0.0, 1.0)  (1.0, 0.0)  (-1.0, 0.0)  (1.0, 0.0)
(-1.0, 0.0)  (0.0, 1.0)  ( 0.0, 1.0)  (0.0, 1.0)
( 1.0, 1.0)  (0.0, 2.0)  (-1.0, 0.0)  (0.0, 1.0)  :End of matrix A
```

10.3 Program Results

```
 F01FMF Example Program Results

 F(A) = EXP(3A)
              1          2          3          4
 1    -10.3264    -1.4883   -12.1206    41.5622
       14.8082    74.3369   -47.0956    32.2927

 2     63.3909   -21.0117    16.5106    -5.1725
      -40.5336   -62.7073    35.2787    17.9413

 3     -6.3954    25.4246   -14.4937   -20.3167
       56.4708    13.8034    -9.2397     2.8647

 4     31.4957    28.6003   -23.8034    23.9841
       23.2757    21.4573   -11.6547    18.7737
```