

# NAG Library Routine Document

## F01FLF

**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

F01FLF 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

```

SUBROUTINE F01FLF (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 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 subroutine 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 Parameters

- 1: N – INTEGER *Input*  
*On entry:*  $n$ , the order of the matrix  $A$ .  
*Constraint:*  $N \geq 0$ .
- 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 F01FLF is called.  
*Constraint:*  $LDA \geq N$ .

4: F – SUBROUTINE, supplied by the user.

*External Procedure*

The subroutine F evaluates  $f(z_i)$  at a number of points  $z_i$ .

The specification of F is:

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

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 F01FLF will terminate the computation, with IFAIL = 2.

2: NZ – INTEGER *Input*

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

3: Z(NZ) – COMPLEX (KIND=nag\_wp) array *Input*

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

4: FZ(NZ) – COMPLEX (KIND=nag\_wp) array *Output*

*On exit:* the  $n_z$  function values.  $FZ(i)$  should return the value  $f(z_i)$ , for  $i = 1, 2, \dots, n_z$ .

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

6: RUSER(\*) – REAL (KIND=nag\_wp) array *User Workspace*

F is called with the parameters IUSER and RUSER as supplied to F01FLF. 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 F01FLF 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 F01FLF, 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 after 40 terms. Further Taylor series coefficients can no longer reliably be obtained by numerical differentiation.

IFAIL = 2

IFLAG has been set nonzero by the user.

IFAIL = 3

The function was unable to compute the Schur decomposition of  $A$ .

**Note:** this failure should not occur and suggests that the routine 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 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  $A^H A = A A^H$ ) 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

F01FLF 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.

F01FLF 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

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 F01FMF can be used to evaluate  $f(A)$  more accurately. If  $A$  is complex Hermitian then it is recommended that F01FFF be used as it is more efficient and, in general, more accurate than F01FLF.

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 F01KBF should be used.

F01ELF 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

```
! F01FLF Example Program Text
! Mark 25 Release. NAG Copyright 2014.
!
! Module f01flfe_mod
!
! F01FLF 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 :: fsin2
```

Contains

```

Subroutine fsin2(iflag,nz,z,fz,iuser,ruser)

!   .. Scalar Arguments ..
Integer, Intent (Inout)           :: iflag
Integer, Intent (In)              :: 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                          :: sin
!   .. Executable Statements ..
Continue
fz(1:nz) = sin((2.0EO_nag_wp,0.0EO_nag_wp)*z(1:nz))
!   Set iflag nonzero to terminate execution for any reason.
iflag = 0
Return
End Subroutine fsin2
End Module f01flfe_mod
Program f01flfe

!   F01FLF Example Main Program

!   .. Use Statements ..
Use nag_library, Only: f01flf, nag_wp, x04daf
Use f01flfe_mod, Only: fsin2
!   .. 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,*) 'F01FLF 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 f01flf(n,a,lda,fsin2,iuser,ruser,iflag,ifail)

!   Print solution
ifail = 0
Call x04daf('G','N',n,n,a,lda,'F(A) = SIN(2A)',ifail)

End Program f01flfe

```

## 10.2 Program Data

F01FLF 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

F01FLF Example Program Results

```

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

```

---