NAG Library Routine Document

F12FDF

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.

Note: *this routine uses* **optional parameters** *to define choices in the problem specification. If you wish to use* default *settings for all of the optional parameters, then this routine need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 for a detailed description of the specification of the optional parameters*.

1 Purpose

F12FDF is an option setting routine in a suite of routines consisting of F12FAF, F12FBF, F12FCF, F12FDF and F12FEF, and may be used to supply individual optional parameters to F12FBF and F12FCF. The initialization routine F12FAF **must** have been called prior to calling F12FDF.

2 Specification

```
SUBROUTINE F12FDF (STR, ICOMM, COMM, IFAIL)
INTEGER            ICOMM(*), IFAIL
REAL (KIND=nag_wp) COMM(*)
CHARACTER(*)       STR
```

3 Description

F12FDF may be used to supply values for optional parameters to F12FBF and F12FCF. It is only necessary to call F12FDF for those arguments whose values are to be different from their default values. One call to F12FDF sets one argument value.

Each optional parameter is defined by a single character string consisting of one or more items. The items associated with a given option must be separated by spaces, or equals signs [=]. Alphabetic characters may be upper or lower case. The string

```
'Pointers = Yes'
```

is an example of a string used to set an optional parameter. For each option the string contains one or more of the following items:

– a mandatory keyword;

– a phrase that qualifies the keyword;

– a number that specifies an integer or real value. Such numbers may be up to 16 contiguous characters in Fortran's I, F, E or D format.

F12FDF does not have an equivalent routine from the ARPACK package which passes options by directly setting values to scalar arguments or to specific elements of array arguments. F12FDF is intended to make the passing of options more transparent and follows the same principle as the single option setting routines in Chapter E04.

The setup routine F12FAF must be called prior to the first call to F12FDF and all calls to F12FDF must precede the first call to F12FBF, the reverse communication iterative solver.

A complete list of optional parameters, their abbreviations, synonyms and default values is given in Section 11.

4 References

Lehoucq R B (2001) Implicitly restarted Arnoldi methods and subspace iteration *SIAM Journal on Matrix Analysis and Applications* **23** 551–562

Lehoucq R B and Scott J A (1996) An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices *Preprint MCS-P547-1195* Argonne National Laboratory

Lehoucq R B and Sorensen D C (1996) Deflation techniques for an implicitly restarted Arnoldi iteration *SIAM Journal on Matrix Analysis and Applications* **17** 789–821

Lehoucq R B, Sorensen D C and Yang C (1998) *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philidelphia

5 Arguments

1: STR – CHARACTER(*) *Input*

On entry: a single valid option string (as described in Section 3 and Section 11).

2: ICOMM(∗) – INTEGER array *Communication Array*

Note: the dimension of the array ICOMM must be at least max(1, LICOMM) (see F12FAF).

On initial entry: must remain unchanged following a call to the setup routine F12FAF.

On exit: contains data on the current options set.

3: COMM(∗) – REAL (KIND=nag_wp) array *Communication Array*

Note: the dimension of the array COMM must be at least 60.

On initial entry: must remain unchanged following a call to the setup routine F12FAF.

On exit: contains data on the current options set.

4: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation 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 argument, 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

The string passed in STR contains an ambiguous keyword.

IFAIL = 2

The string passed in STR contains a keyword that could not be recognized.

IFAIL $= 3$

The string passed in STR contains a second keyword that could not be recognized.

IFAIL $= 4$

The initialization routine F12FAF has not been called or a communication array has become corrupted.

IFAIL $= -99$

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

F12FDF is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example solves $Ax = \lambda Bx$ in **Shifted Inverse** mode, where A and B are obtained from the standard central difference discretization of the one-dimensional Laplacian operator $\frac{\partial^2 u}{\partial x^2}$ on $[0, 1]$, with zero Dirichlet boundary conditions. Data is passed to and from the reverse communication routine F12FBF using pointers to the communication array.

10.1 Program Text

```
!   F12FDF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module f12fdfe_mod

!     F12FDF 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                           :: mv
!     .. Parameters ..
```

```
      Real (Kind=nag_wp), Parameter, Public :: four = 4.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: six = 6.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Integer, Parameter, Public       :: imon = 0, ipoint = 1, licomm = 140,  &
                                          nin = 5, nout = 6
    Contains
      Subroutine mv(n,v,w)

!       .. Use Statements ..
        Use nag_library, Only: dscal
!       .. Scalar Arguments ..
        Integer, Intent (In)          :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: v(n)
        Real (Kind=nag_wp), Intent (Out) :: w(n)
!       .. Local Scalars ..
        Real (Kind=nag_wp)            :: h
        Integer                       :: j
!       .. Intrinsic Procedures ..
        Intrinsic                     :: real
!       .. Executable Statements ..
        h = one/(real(n+1,kind=nag_wp)*six)
        w(1) = four*v(1) + v(2)
        Do j = 2, n - 1
          w(j) = v(j-1) + four*v(j) + v(j+1)
        End Do
        j = n
        w(j) = v(j-1) + four*v(j)
!       The NAG name equivalent of dscal is f06edf
        Call dscal(n,h,w,1)
        Return
      End Subroutine mv
    End Module f12fdfe_mod
    Program f12fdfe

!     F12FDF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: dcopy, dgttrf, dgttrs, dnrm2, f12faf, f12fbf,     &
                             f12fcf, f12fdf, f12fef, nag_wp
      Use f12fdfe_mod, Only: four, imon, ipoint, licomm, mv, nin, nout, one,   &
                             six, two, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: h, r1, r2, sigma
      Integer                         :: ifail, info, irevcm, j, lcomm, ldv,  &
                                         n, nconv, ncv, nev, niter, nshift
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: ad(:), adl(:), adu(:), adu2(:),      &
                                          comm(:), d(:,:), mx(:), resid(:),    &
                                          v(:,:), x(:)
      Integer                         :: icomm(licomm)
      Integer, Allocatable            :: ipiv(:)
!     .. Intrinsic Procedures ..
      Intrinsic                       :: real
!     .. Executable Statements ..
      Write (nout,*) 'F12FDF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) n, nev, ncv

      lcomm = 3*n + ncv*ncv + 8*ncv + 60
      ldv = n
      Allocate (ad(n),adl(n),adu(n),adu2(n),comm(lcomm),d(ncv,2),mx(n),        &
        resid(n),v(ldv,ncv),x(n),ipiv(n))

      ifail = 0
```

```
      Call f12faf(n,nev,ncv,icomm,licomm,comm,lcomm,ifail)

!     We are solving a generalized problem
      ifail = 0
      Call f12fdf('GENERALIZED',icomm,comm,ifail)
!     Indicate that we are using the shift and invert mode.
      Call f12fdf('SHIFTED INVERSE',icomm,comm,ifail)
      If (ipoint==1) Then
!       Use pointers to Workspace in calculating matrix vector
!       products rather than interfacing through the array X
        Call f12fdf('POINTERS=YES',icomm,comm,ifail)
      End If

      h = one/real(n+1,kind=nag_wp)
      r1 = (four/six)*h
      r2 = (one/six)*h
      sigma = zero
      ad(1:n) = two/h - sigma*r1
      adl(1:n) = -one/h - sigma*r2
      adu(1:n) = adl(1:n)
!     The NAG name equivalent of dgttrf is f07cdf
      Call dgttrf(n,adl,ad,adu,adu2,ipiv,info)

      irevcm = 0
      ifail = -1
revcm: Do
        Call f12fbf(irevcm,resid,v,ldv,x,mx,nshift,comm,icomm,ifail)
        If (irevcm==5) Then
          Exit revcm
        Else If (irevcm==-1) Then
!         Perform  y <--- OP*x = inv[A-SIGMA*M]*M*x
!         The NAG name equivalent of dgttrs is f07cef
          If (ipoint==0) Then
            Call mv(n,x,mx)
            x(1:n) = mx(1:n)
            Call dgttrs('N',n,1,adl,ad,adu,adu2,ipiv,x,n,info)
          Else
            Call mv(n,comm(icomm(1)),comm(icomm(2)))
            Call dgttrs('N',n,1,adl,ad,adu,adu2,ipiv,comm(icomm(2)),n,info)
          End If
        Else If (irevcm==1) Then
!         Perform y <-- OP*x = inv[A-sigma*M]*M*x
!         The NAG name equivalent of dgttrs is f07cef.
!         M*x has been saved in COMM(ICOMM(3)) or MX.
          If (ipoint==0) Then
            x(1:n) = mx(1:n)
            Call dgttrs('N',n,1,adl,ad,adu,adu2,ipiv,x,n,info)
          Else
!           The NAG name equivalent of dcopy is f06eff
            Call dcopy(n,comm(icomm(3)),1,comm(icomm(2)),1)
            Call dgttrs('N',n,1,adl,ad,adu,adu2,ipiv,comm(icomm(2)),n,info)
          End If
        Else If (irevcm==2) Then
!         Perform  y <--- M*x.
          If (ipoint==0) Then
            Call mv(n,x,mx)
          Else
            Call mv(n,comm(icomm(1)),comm(icomm(2)))
          End If
        Else If (irevcm==4 .And. imon/=0) Then
!         Output monitoring information
          Call f12fef(niter,nconv,d,d(1,2),icomm,comm)
!         The NAG name equivalent of dnrm2 is f06ejf
          Write (6,99999) niter, nconv, dnrm2(nev,d(1,2),1)
        End If
      End Do revcm

      If (ifail==0) Then
!       Post-Process using F12FCF to compute eigenvalues/values.
        ifail = 0
        Call f12fcf(nconv,d,v,ldv,sigma,resid,v,ldv,comm,icomm,ifail)
```

```
        Write (nout,99998) nconv, sigma
        Write (nout,99997)(j,d(j,1),j=1,nconv)
      End If

99999 Format (1X,'Iteration',1X,I3,', No. converged =',1X,I3,', norm o',       &
        'f estimates =',E16.8)
99998 Format (1X,/,' The ',I4,' Ritz values of closest to ',F8.4,' are:',/)
99997 Format (1X,I8,5X,F12.4)
    End Program f12fdfe
```

10.2 Program Data

```
F12FDF Example Program Data
 100 4 10 : Values for N NEV and NCV
```

10.3 Program Results

```
 F12FDF Example Program Results


 The    4 Ritz values of closest to   0.0000 are:

        1           9.8704
        2          39.4912
        3          88.8909
        4         158.1175
```

11 Optional Parameters

Several optional parameters for the computational routines F12FBF and F12FCF define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of F12FBF and F12FCF these optional parameters have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 11.1.

Advisory
Both Ends
Buckling
Cayley
Defaults
Exact Shifts
Generalized
Initial Residual
Iteration Limit
Largest Algebraic
Largest Magnitude
List
Monitoring
Nolist
Pointers
Print Level
Random Residual
Regular

Regular Inverse
Shifted Inverse
Smallest Algebraic
Smallest Magnitude
Standard
Supplied Shifts
Tolerance
Vectors

Optional parameters may be specified by calling F12FDF before a call to F12FBF, but after a call to F12FAF. One call is necessary for each optional parameter.

All optional parameters you do not specify are set to their default values. Optional parameters you do specify are unaltered by F12FBF and F12FCF (unless they define invalid values) and so remain in effect for subsequent calls unless you alter them.

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value, where the symbol ϵ is a generic notation for ***machine precision*** (see X02AJF).

Keywords and character values are case and white space insensitive.

Advisory i Default $=$ the value returned by X04ABF

The destination for advisory messages.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Exact Shifts Default
Supplied Shifts

During the Lanczos iterative process, shifts are applied internally as part of the implicit restarting scheme. The shift strategy used by default and selected by the **Exact Shifts** is strongly recommended over the alternative **Supplied Shifts** (see Lehoucq *et al.* (1998) for details of shift strategies).

If **Exact Shifts** are used then these are computed internally by the algorithm in the implicit restarting scheme.

If **Supplied Shifts** are used then, during the Lanczos iterative process, you must supply shifts through array arguments of F12FBF when F12FBF returns with IREVCM $= 3$; the real and imaginary parts of the shifts are returned in X and MX respectively (or in COMM when the option **Pointers** $=$ YES is set). This option should only be used if you are an experienced user since this requires some algorithmic knowledge and because more operations are usually required than for the implicit shift scheme. Details on the use of explicit shifts and further references on shift strategies are available in Lehoucq *et al.* (1998).

Iteration Limit i Default $= 300$

The limit on the number of Lanczos iterations that can be performed before F12FBF exits. If not all requested eigenvalues have converged to within **Tolerance** and the number of Lanczos iterations has reached this limit then F12FBF exits with an error; F12FCF can still be called subsequently to return

the number of converged eigenvalues, the converged eigenvalues and, if requested, the corresponding eigenvectors.

Largest Magnitude Default
Both Ends
Largest Algebraic
Smallest Algebraic
Smallest Magnitude

The Lanczos iterative method converges on a number of eigenvalues with given properties. The default is for F12FBF to compute the eigenvalues of largest magnitude using **Largest Magnitude**. Alternatively, eigenvalues may be chosen which have **Largest Algebraic** part, **Smallest Magnitude**, or **Smallest Algebraic** part; or eigenvalues which are from **Both Ends** of the algebraic spectrum.

Note that these options select the eigenvalue properties for eigenvalues of OP (and B for **Generalized** problems), the linear operator determined by the computational mode and problem type.

Nolist Default
List

Normally each optional parameter specification is not printed to the advisory channel as it is supplied. Optional parameter **List** may be used to enable printing and optional parameter **Nolist** may be used to suppress the printing.

Monitoring i Default $= -1$

If $i > 0$, monitoring information is output to channel number i during the solution of each problem; this may be the same as the **Advisory** channel number. The type of information produced is dependent on the value of **Print Level**, see the description of the optional parameter **Print Level** for details of the information produced. Please see X04ACF to associate a file with a given channel number.

Pointers Default $=$ NO

During the iterative process and reverse communication calls to F12FBF, required data can be communicated to and from F12FBF in one of two ways. When **Pointers** $=$ NO is selected (the default) then the array arguments X and MX are used to supply you with required data and used to return computed values back to F12FBF. For example, when IREVCM $= 1$, F12FBF returns the vector x in X and the matrix-vector product Bx in MX and expects the result or the linear operation OP(x) to be returned in X.

If **Pointers** $=$ YES is selected then the data is passed through sections of the array argument COMM. The section corresponding to X when **Pointers** $=$ NO begins at a location given by the first element of ICOMM; similarly the section corresponding to MX begins at a location given by the second element of ICOMM. This option allows F12FBF to perform fewer copy operations on each intermediate exit and entry, but can also lead to less elegant code in the calling program.

Print Level i Default $= 0$

This controls the amount of printing produced by F12FDF as follows.

$= 0$ No output except error messages. If you want to suppress all output, set **Print Level** $= 0$.

> 0 The set of selected options.

$= 2$ Problem and timing statistics on final exit from F12FBF.

≥ 5 A single line of summary output at each Lanczos iteration.

≥ 10 If **Monitoring** > 0, **Monitoring** is set, then at each iteration, the length and additional steps of the current Lanczos factorization and the number of converged Ritz values; during re-orthogonalization, the norm of initial/restarted starting vector; on a final Lanczos iteration, the number of update iterations taken, the number of converged eigenvalues, the converged eigenvalues and their Ritz estimates.

≥ 20 Problem and timing statistics on final exit from F12FBF. If **Monitoring** > 0, **Monitoring** is set, then at each iteration, the number of shifts being applied, the eigenvalues and estimates of the symmetric tridiagonal matrix H, the size of the Lanczos basis, the wanted Ritz values and associated Ritz estimates and the shifts applied; vector norms prior to and following re-orthogonalization.

≥ 30 If **Monitoring** > 0, **Monitoring** is set, then on final iteration, the norm of the residual; when computing the Schur form, the eigenvalues and Ritz estimates both before and after sorting; for each iteration, the norm of residual for compressed factorization and the symmetric tridiagonal matrix H; during re-orthogonalization, the initial/restarted starting vector; during the Lanczos iteration loop, a restart is flagged and the number of the residual requiring iterative refinement; while applying shifts, some indices.

≥ 40 If **Monitoring** > 0, **Monitoring** is set, then during the Lanczos iteration loop, the Lanczos vector number and norm of the current residual; while applying shifts, key measures of progress and the order of H; while computing eigenvalues of H, the last rows of the Schur and eigenvector matrices; when computing implicit shifts, the eigenvalues and Ritz estimates of H.

≥ 50 If **Monitoring** is set, then during Lanczos iteration loop: norms of key components and the active column of H, norms of residuals during iterative refinement, the final symmetric tridiagonal matrix H; while applying shifts: number of shifts, shift values, block indices, updated tridiagonal matrix H; while computing eigenvalues of H: the diagonals of H, the computed eigenvalues and Ritz estimates.

Note that setting **Print Level** ≥ 30 can result in very lengthy **Monitoring** output.

Note that setting **Print Level** ≥ 30 can result in very lengthy **Monitoring** output.

Random Residual Default
Initial Residual

To begin the Lanczos iterative process, F12FBF requires an initial residual vector. By default F12FBF provides its own random initial residual vector; this option can also be set using optional parameter **Random Residual**. Alternatively, you can supply an initial residual vector (perhaps from a previous computation) to F12FBF through the array argument RESID; this option can be set using optional parameter **Initial Residual**.

Regular Default
Regular Inverse
Shifted Inverse
Buckling
Cayley

These options define the computational mode which in turn defines the form of operation OP(x) to be performed when F12FBF returns with IREVCM $= -1$ or 1 and the matrix-vector product Bx when F12FBF returns with IREVCM $= 2$.

Given a **Standard** eigenvalue problem in the form $Ax = \lambda x$ then the following modes are available with the appropriate operator OP(x).

Regular	OP $= A$
Shifted Inverse	OP $= (A - \sigma I)^{-1}$ where σ is real

Given a **Generalized** eigenvalue problem in the form $Ax = \lambda Bx$ then the following modes are available with the appropriate operator OP(x).

Regular Inverse	OP $= B^{-1}A$
Shifted Inverse	OP $= (A - \sigma B)^{-1}B$, where σ is real
Buckling	OP $= (B - \sigma A)^{-1}A$, where σ is real
Cayley	OP $= (A - \sigma B)^{-1}(A + \sigma B)$, where σ is real

Standard Default
Generalized

The problem to be solved is either a standard eigenvalue problem, $Ax = \lambda x$, or a generalized eigenvalue problem, $Ax = \lambda Bx$. The optional parameter **Standard** should be used when a standard eigenvalue problem is being solved and the optional parameter **Generalized** should be used when a generalized eigenvalue problem is being solved.

Tolerance r Default $= \epsilon$

An approximate eigenvalue has deemed to have converged when the corresponding Ritz estimate is within **Tolerance** relative to the magnitude of the eigenvalue.

Vectors Default = RITZ

The routine F12FCF can optionally compute the Schur vectors and/or the eigenvectors corresponding to the converged eigenvalues. To turn off computation of any vectors the option **Vectors** = NONE should be set. To compute only the Schur vectors (at very little extra cost), the option **Vectors** = SCHUR should be set and these will be returned in the array argument V of F12FCF. To compute the eigenvectors (Ritz vectors) corresponding to the eigenvalue estimates, the option **Vectors** = RITZ should be set and these will be returned in the array argument Z of F12FCF, if Z is set equal to V (as in Section 10) then the Schur vectors in V are overwritten by the eigenvectors computed by F12FCF.