# NAG Library Routine Document

# F12FAF

**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

F12FAF is a setup routine in a suite of routines consisting of F12FAF, F12FBF, F12FCF, F12FDF and F12FEF. It is used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real symmetric matrices.

The suite of routines is suitable for the solution of large sparse, standard or generalized, symmetric eigenproblems where only a few eigenvalues from a selected range of the spectrum are required.

## 2 Specification

```
SUBROUTINE F12FAF (N, NEV, NCV, ICOMM, LICOMM, COMM, LCOMM, IFAIL)
INTEGER            N, NEV, NCV, ICOMM(max(1,LICOMM)), LICOMM, LCOMM,       &
                   IFAIL
REAL (KIND=nag_wp) COMM(max(1,LCOMM))
```

## 3 Description

The suite of routines is designed to calculate some of the eigenvalues, $\lambda$, (and optionally the corresponding eigenvectors, $x$) of a standard eigenvalue problem $Ax = \lambda x$, or of a generalized eigenvalue problem $Ax = \lambda Bx$ of order $n$, where $n$ is large and the coefficient matrices $A$ and $B$ are sparse, real and symmetric. The suite can also be used to find selected eigenvalues/eigenvectors of smaller scale dense, real and symmetric problems.

F12FAF is a setup routine which must be called before F12FBF, the reverse communication iterative solver, and before F12FDF, the options setting routine. F12FCF, is a post-processing routine that must be called following a successful final exit from F12FBF, while F12FEF can be used to return additional monitoring information during the computation.

This setup routine initializes the communication arrays, sets (to their default values) all options that can be set by you via the option setting routine F12FDF, and checks that the lengths of the communication arrays as passed by you are of sufficient length. For details of the options available and how to set them see Section 11.1 in F12FDF.

## 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 Parameters

1: N – INTEGER *Input*

*On entry*: the order of the matrix $A$ (and the order of the matrix $B$ for the generalized problem) that defines the eigenvalue problem.

*Constraint*: $\mathrm{N} > 0$.

2: NEV – INTEGER *Input*

*On entry*: the number of eigenvalues to be computed.

*Constraint*: $0 < \mathrm{NEV} < \mathrm{N} - 1$.

3: NCV – INTEGER *Input*

*On entry*: the number of Lanczos basis vectors to use during the computation.

At present there is no *a priori* analysis to guide the selection of NCV relative to NEV. However, it is recommended that $\mathrm{NCV} \geq 2 \times \mathrm{NEV} + 1$. If many problems of the same type are to be solved, you should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of matrix-vector operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal 'cross-over' with respect to CPU time is problem dependent and must be determined empirically.

*Constraint*: $\mathrm{NEV} < \mathrm{NCV} \leq \mathrm{N}$.

4: ICOMM(max(1, LICOMM)) – INTEGER array *Communication Array*

*On exit*: contains data to be communicated to the other routines in the suite.

5: LICOMM – INTEGER *Input*

*On entry*: the dimension of the array ICOMM as declared in the (sub)program from which F12FAF is called.

If $\mathrm{LICOMM} = -1$, a workspace query is assumed and the routine only calculates the required dimensions of ICOMM and COMM, which it returns in $\mathrm{ICOMM}(1)$ and $\mathrm{COMM}(1)$ respectively.

*Constraint*: $\mathrm{LICOMM} \geq 140$ or $\mathrm{LICOMM} = -1$.

6: COMM(max(1, LCOMM)) – REAL (KIND=nag_wp) array *Communication Array*

*On exit*: contains data to be communicated to the other routines in the suite.

7: LCOMM – INTEGER *Input*

*On entry*: the dimension of the array COMM as declared in the (sub)program from which F12FAF is called.

If $\mathrm{LCOMM} = -1$, a workspace query is assumed and the routine only calculates the dimensions of ICOMM and COMM required by F12FBF, which it returns in $\mathrm{ICOMM}(1)$ and $\mathrm{COMM}(1)$ respectively.

*Constraint*: $\mathrm{LCOMM} \geq 3 \times \mathrm{N} + \mathrm{NCV} \times \mathrm{NCV} + 8 \times \mathrm{NCV} + 60$ or $\mathrm{LCOMM} = -1$.

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 $= 1$

On entry, N $\leq 0$.

IFAIL $= 2$

On entry, NEV $\leq 0$.

IFAIL $= 3$

On entry, NCV $\leq$ NEV or NCV $>$ N.

IFAIL $= 4$

On entry, LICOMM $< 140$ and LICOMM $\neq -1$.

IFAIL $= 5$

On entry, LCOMM $< 3 \times$ N $+$ NCV $\times$ NCV $+ 8 \times$ NCV $+ 60$ and LCOMM $\neq -1$.

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

Not applicable.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

None.

# 10 Example

This example solves $Ax = \lambda x$ in regular mode, where $A$ is obtained from the standard central difference discretization of the Laplacian operator $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ on the unit square, with zero Dirichlet boundary conditions. Eigenvalues of smallest magnitude are selected.

## 10.1 Program Text

```
!   F12FAF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module f12fafe_mod

!     F12FAF 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                                :: av
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter         :: one = 1.0_nag_wp
      Integer, Parameter, Public            :: imon = 0, ipoint = 0,            &
                                               licomm = 140, nin = 5, nout = 6
    Contains
      Subroutine tv(nx,x,y)
!       Compute the matrix vector multiplication y<---T*x where T is a nx
!       by nx tridiagonal matrix.

!       .. Scalar Arguments ..
        Integer, Intent (In)                :: nx
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: x(nx)
        Real (Kind=nag_wp), Intent (Out)    :: y(nx)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: dd, dl, du
        Integer                             :: j
!       .. Executable Statements ..
        dd = 4.0_nag_wp
        dl = -one
        du = -one
        y(1) = dd*x(1) + du*x(2)
        Do j = 2, nx - 1
          y(j) = dl*x(j-1) + dd*x(j) + du*x(j+1)
        End Do
        y(nx) = dl*x(nx-1) + dd*x(nx)
        Return
      End Subroutine tv
      Subroutine av(nx,v,w)

!       .. Use Statements ..
        Use nag_library, Only: daxpy, dscal
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: nx
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: v(nx*nx)
        Real (Kind=nag_wp), Intent (Out)    :: w(nx*nx)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: h2
        Integer                             :: j, lo, n
!       .. Intrinsic Procedures ..
        Intrinsic                           :: real
!       .. Executable Statements ..
        h2 = one/real((nx+1)*(nx+1),kind=nag_wp)
        Call tv(nx,v(1),w(1))
!       The NAG name equivalent of daxpy is f06ecf
```

```
        Call daxpy(nx,-one,v(nx+1),1,w(1),1)
        Do j = 2, nx - 1
          lo = (j-1)*nx
          Call tv(nx,v(lo+1),w(lo+1))
          Call daxpy(nx,-one,v(lo-nx+1),1,w(lo+1),1)
          Call daxpy(nx,-one,v(lo+nx+1),1,w(lo+1),1)
        End Do
        lo = (nx-1)*nx
        Call tv(nx,v(lo+1),w(lo+1))
        Call daxpy(nx,-one,v(lo-nx+1),1,w(lo+1),1)
        n = nx*nx
!       The NAG name equivalent of dscal is f06edf
        Call dscal(n,one/h2,w,1)
        Return
      End Subroutine av
    End Module f12fafe_mod
    Program f12fafe

!     F12FAF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: dnrm2, f12faf, f12fbf, f12fcf, f12fdf, f12fef,    &
                             nag_wp
      Use f12fafe_mod, Only: av, imon, ipoint, licomm, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: sigma
      Integer                             :: ifail, irevcm, j, lcomm, ldv, n, &
                                             nconv, ncv, nev, niter, nshift, nx
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: ax(:), comm(:), d(:,:), mx(:),   &
                                             resid(:), v(:,:), x(:)
      Integer                             :: icomm(licomm)
!     .. Executable Statements ..
      Write (nout,*) 'F12FAF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) nx, nev, ncv

      n = nx*nx
      ldv = n
      lcomm = 3*n + ncv*ncv + 8*ncv + 60
      Allocate (ax(n),comm(lcomm),d(ncv,2),mx(n),resid(n),v(ldv,ncv),x(n))

      ifail = 0
      Call f12faf(n,nev,ncv,icomm,licomm,comm,lcomm,ifail)

      ifail = 0
      Call f12fdf('SMALLEST MAGNITUDE',icomm,comm,ifail)
!     Increase the iteration limit if required.
      Call f12fdf('ITERATION LIMIT=500',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

      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 .Or. irevcm==1) Then
!         Perform matrix vector multiplication y <--- Op*x
          If (ipoint==0) Then
            Call av(nx,x,ax)
            x(1:n) = ax(1:n)
          Else
```

```
            Call av(nx,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 eigenvalue/vectors.
        ifail = 0
        Call f12fcf(nconv,d,v,ldv,sigma,resid,v,ldv,comm,icomm,ifail)
        Write (nout,99998) nconv
        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 smallest magnitude are:'/)
99997 Format (1X,I8,5X,F12.4)
    End Program f12fafe
```

## 10.2 Program Data

```
F12FAF Example Program Data
 10 4 10 : Values for NX NEV and NCV
```

## 10.3 Program Results

```
 F12FAF Example Program Results


 The    4 Ritz values of smallest magnitude are:

        1          19.6054
        2          48.2193
        3          48.2193
        4          76.8333
```