NAG Library Routine Document

F12AAF

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

F12AAF is a setup routine in a suite of routines consisting of F12AAF, F12ABF, F12ACF, F12ADF and F12AEF. It is used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices.

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

2 Specification

```
SUBROUTINE F12AAF (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, λ, (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 nonsymmetric. The suite can also be used to find selected eigenvalues/eigenvectors of smaller scale dense, real and nonsymmetric problems.

F12AAF is a setup routine which must be called before F12ABF, the reverse communication iterative solver, and before F12ADF, the options setting routine. F12ACF is a post-processing routine that must be called following a successful final exit from F12ABF, while F12AEF 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 F12ADF, 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 F12ADF.

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: 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 Arnoldi 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: $\mathrm{NEV} + 1 < \mathrm{NCV} \leq \mathrm{N}$.

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 F12AAF 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 F12AAF is called.

If $\mathrm{LCOMM} = -1$, a workspace query is assumed and the routine only calculates the dimensions of ICOMM and COMM required by F12ABF, which it returns in $\mathrm{ICOMM}(1)$ and $\mathrm{COMM}(1)$ respectively.

Constraint: $\mathrm{LCOMM} \geq 3 \times \mathrm{N} + 3 \times \mathrm{NCV} \times \mathrm{NCV} + 6 \times \mathrm{NCV} + 60$ 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 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 $= 1$

On entry, N ≤ 0.

IFAIL $= 2$

On entry, NEV ≤ 0.

IFAIL $= 3$

On entry, NCV $<$ NEV $+ 2$ or NCV $>$ N.

IFAIL $= 4$

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

IFAIL $= 5$

On entry, LCOMM $< 3 \times$ N $+ 3 \times$ NCV $\times$ NCV $+ 6 \times$ NCV $+ 60$ and LCOMM $\neq -1$.

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

F12AAF is not threaded in any implementation.

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 convection-diffusion operator $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \rho\frac{\partial u}{\partial x}$ on the unit square, with zero Dirichlet boundary conditions, where $\rho = 100$.

10.1 Program Text

```
!   F12AAF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module f12aafe_mod

!     F12AAF 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 ..
      Integer, Parameter, Public         :: imon = 0, ipoint = 0, 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 with constant diagonals (DD, DL and DU).

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: half = 0.5_nag_wp
        Real (Kind=nag_wp), Parameter  :: rho = 100.0_nag_wp
!       .. 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, nx1, nx2
        Integer                        :: j
!       .. Intrinsic Procedures ..
        Intrinsic                      :: real
!       .. Executable Statements ..
        nx1 = real(nx+1,kind=nag_wp)
        nx2 = nx1*nx1
        dd = 4.0_nag_wp*nx2
        dl = -nx2 - half*rho*nx1
        du = -nx2 + half*rho*nx1
        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
!       .. 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)             :: nx2
        Integer                        :: j, lo
```

```
!       .. Intrinsic Procedures ..
        Intrinsic                      :: real
!       .. Executable Statements ..
        nx2 = -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,nx2,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,nx2,v(lo-nx+1),1,w(lo+1),1)
          Call daxpy(nx,nx2,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,nx2,v(lo-nx+1),1,w(lo+1),1)
        Return
      End Subroutine av
    End Module f12aafe_mod
    Program f12aafe

!     F12AAF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: dnrm2, f12aaf, f12abf, f12acf, f12adf, f12aef,    &
                             nag_wp
      Use f12aafe_mod, Only: av, imon, ipoint, nin, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: sigmai, sigmar
      Integer                          :: i, ifail, ifail1, irevcm, lcomm,     &
                                          ldv, licomm, n, nconv, ncv, nev,     &
                                          niter, nshift, nx
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: ax(:), comm(:), d(:,:), mx(:),       &
                                          resid(:), v(:,:), x(:)
      Integer, Allocatable             :: icomm(:)
!     .. Executable Statements ..
      Write (nout,*) 'F12AAF 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 + 3*ncv*ncv + 6*ncv + 60
      licomm = 140
      Allocate (ax(n),comm(lcomm),d(ncv,3),mx(n),resid(n),v(ldv,ncv),x(n),     &
        icomm(licomm))

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call f12aaf(n,nev,ncv,icomm,licomm,comm,lcomm,ifail)

!     Set the region of the spectrum that is required.
      ifail = 0
      Call f12adf('SMALLEST MAG',icomm,comm,ifail)

      If (ipoint/=0) Then

!       Use pointers to workspace in calculating matrix vector products
!       rather than interfacing through the array X.
        ifail = 0
        Call f12adf('POINTERS=YES',icomm,comm,ifail)

      End If

      irevcm = 0
      ifail = -1
```

```
loop: Do
        Call f12abf(irevcm,resid,v,ldv,x,mx,nshift,comm,icomm,ifail)

        If (irevcm/=5) Then
          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

!           Set IMON=1 to output monitoring information.
            Call f12aef(niter,nconv,d,d(1,2),d(1,3),icomm,comm)

!           The NAG name equivalent of dnrm2 is f06ejf
            Write (6,99999) niter, nconv, dnrm2(nev,d(1,3),1)
          End If
        Else
          Exit loop
        End If
      End Do loop

      If (ifail==0) Then

!       Post-Process using F12ACF to compute eigenvalues and
!       (by default) the corresponding eigenvectors.
        ifail1 = 0
        Call f12acf(nconv,d,d(1,2),v,ldv,sigmar,sigmai,resid,v,ldv,comm,icomm, &
          ifail1)

        Write (nout,99998) nconv
        Do i = 1, nconv
          Write (nout,99997) i, d(i,1), d(i,2)
        End Do
      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,' , ',F12.4,' )')
    End Program f12aafe
```

10.2 Program Data

```
F12AAF Example Program Data
 10 10 30 : Values for NX NEV and NCV
```

10.3 Program Results

```
 F12AAF Example Program Results


 The   10 Ritz values of smallest magnitude are:

        1     (     251.8027 ,     152.7109 )
        2     (     251.8027 ,    -152.7109 )
        3     (     280.4166 ,     152.7109 )
        4     (     280.4166 ,    -152.7109 )
        5     (     325.5237 ,     152.7109 )
```

```
       6     (     325.5237 ,    -152.7109 )
       7     (     383.4696 ,     152.7109 )
       8     (     383.4696 ,    -152.7109 )
       9     (     449.5598 ,     152.7109 )
      10     (     449.5598 ,    -152.7109 )
```