NAG Library Routine Document

F12ABF

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 the option setting routine F12ADF need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 in F12ADF for a detailed description of the specification of the optional parameters.*

1 Purpose

F12ABF is an iterative solver used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices. This is part of a suite of routines that also includes F12AAF, F12ACF, F12ADF and F12AEF. It is

2 Specification

```
SUBROUTINE F12ABF (IREVCM, RESID, V, LDV, X, MX, NSHIFT, COMM, ICOMM,      &
                   IFAIL)
INTEGER            IREVCM, LDV, NSHIFT, ICOMM(*), IFAIL
REAL (KIND=nag_wp) RESID(*), V(LDV,*), X(*), MX(*), COMM(*)
```

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.

F12ABF is a **reverse communication** routine, based on the ARPACK routine **dnaupd**, using the Implicitly Restarted Arnoldi iteration method. The method is described in Lehoucq and Sorensen (1996) and Lehoucq (2001) while its use within the ARPACK software is described in great detail in Lehoucq *et al.* (1998). An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices is provided in Lehoucq and Scott (1996). This suite of routines offers the same functionality as the ARPACK software for real nonsymmetric problems, but the interface design is quite different in order to make the option setting clearer and to simplify the interface of F12ABF.

The setup routine F12AAF must be called before F12ABF, the reverse communication iterative solver. Options may be set for F12ABF by prior calls to the option setting routine F12ADF and a post-processing routine F12ACF must be called following a successful final exit from F12ABF. F12AEF, may be called following certain flagged, intermediate exits from F12ABF to provide additional monitoring information about the computation.

F12ABF uses **reverse communication**, i.e., it returns repeatedly to the calling program with the argument IREVCM (see Section 5) set to specified values which require the calling program to carry out one of the following tasks:

– compute the matrix-vector product $y = \mathrm{OP}x$, where OP is defined by the computational mode;

– compute the matrix-vector product $y = Bx$;

– notify the completion of the computation;

– allow the calling program to monitor the solution.

The problem type to be solved (standard or generalized), the spectrum of eigenvalues of interest, the mode used (regular, regular inverse, shifted inverse, shifted real or shifted imaginary) and other options

can all be set using the option setting routine F12ADF (see Section 11.1 in F12ADF for details on setting options and of the default settings).

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

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **IREVCM**. Between intermediate exits and re-entries, **all arguments other than X, MX and COMM must remain unchanged**.

1: IREVCM – INTEGER *Input/Output*

On initial entry: IREVCM $= 0$, otherwise an error condition will be raised.

On intermediate re-entry: must be unchanged from its previous exit value. Changing IREVCM to any other value between calls will result in an error.

On intermediate exit: has the following meanings.

IREVCM $= -1$
The calling program must compute the matrix-vector product $y = \mathrm{OP}x$, where x is stored in X (by default) or in the array COMM (starting from the location given by the first element of ICOMM) when the option **Pointers** = YES is set in a prior call to F12ADF. The result y is returned in X (by default) or in the array COMM (starting from the location given by the second element of ICOMM) when the option **Pointers** = YES is set. If B is not symmetric semidefinite then the precomputed values in MX should not be used (see the explanation under IREVCM $= 2$).

IREVCM $= 1$
The calling program must compute the matrix-vector product $y = \mathrm{OP}x$. This is similar to the case IREVCM $= -1$ except that the result of the matrix-vector product Bx (as required in some computational modes) has already been computed and is available in MX (by default) or in the array COMM (starting from the location given by the third element of ICOMM) when the option **Pointers** = YES is set.

IREVCM $= 2$
The calling program must compute the matrix-vector product $y = Bx$, where x is stored as described in the case IREVCM $= -1$ and y is returned in the location described by the case IREVCM $= 1$. This computation is requested when solving the **Generalized** problem using either **Shifted Inverse Imaginary** or **Shifted Inverse Real**; in these cases B is used as an inner-product space and requires that B be symmetric semidefinite. If neither A nor B is symmetric semidefinite then the problem should be reformulated in a **Standard** form.

IREVCM $= 3$
Compute the NSHIFT real and imaginary parts of the shifts where the real parts are to be returned in the first NSHIFT locations of the array X and the imaginary parts are to be returned in the first NSHIFT locations of the array MX. Only complex conjugate pairs of shifts may be applied and the pairs must be placed in consecutive locations. This value of IREVCM will only arise if the optional parameter **Supplied Shifts** is set in a prior call to F12ADF which is intended for experienced users only; the default and recommended option is to use exact shifts (see Lehoucq *et al.* (1998) for details).

IREVCM = 4
Monitoring step: a call to F12AEF can now be made to return the number of Arnoldi iterations, the number of converged Ritz values, their real and imaginary parts, and the corresponding Ritz estimates.

On final exit: IREVCM = 5: F12ABF has completed its tasks. The value of IFAIL determines whether the iteration has been successfully completed, or whether errors have been detected. On successful completion F12ACF must be called to return the requested eigenvalues and eigenvectors (and/or Schur vectors).

Constraint: on initial entry, IREVCM = 0; on re-entry IREVCM must remain unchanged.

2: RESID(∗) – REAL (KIND=nag_wp) array *Input/Output*

Note: the dimension of the array RESID must be at least N (see F12AAF).

On initial entry: need not be set unless the option **Initial Residual** has been set in a prior call to F12ADF in which case RESID should contain an initial residual vector, possibly from a previous run.

On intermediate re-entry: must be unchanged from its previous exit. Changing RESID to any other value between calls may result in an error exit.

On intermediate exit: contains the current residual vector.

On final exit: contains the final residual vector.

3: V(LDV, ∗) – REAL (KIND=nag_wp) array *Input/Output*

Note: the second dimension of the array V must be at least max(1, NCV) (see F12AAF).

On initial entry: need not be set.

On intermediate re-entry: must be unchanged from its previous exit.

On intermediate exit: contains the current set of Arnoldi basis vectors.

On final exit: contains the final set of Arnoldi basis vectors.

4: LDV – INTEGER *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which F12ABF is called.

Constraint: LDV ≥ N.

5: X(∗) – REAL (KIND=nag_wp) array *Input/Output*

Note: the dimension of the array X must be at least N if **Pointers** = NO (default) and at least 1 if **Pointers** = YES (see F12AAF).

On initial entry: need not be set, it is used as a convenient mechanism for accessing elements of COMM.

On intermediate re-entry: if **Pointers** = YES, X need not be set.

If **Pointers** = NO, X must contain the result of $y = \mathrm{OP}x$ when IREVCM returns the value -1 or $+1$. It must return the real parts of the computed shifts when IREVCM returns the value 3.

On intermediate exit: if **Pointers** = YES, X is not referenced.

If **Pointers** = NO, X contains the vector x when IREVCM returns the value -1 or $+1$.

On final exit: does not contain useful data.

6: MX(∗) – REAL (KIND=nag_wp) array *Input/Output*

Note: the dimension of the array MX must be at least N if **Pointers** = NO (default) and at least 1 if **Pointers** = YES (see F12AAF).

On initial entry: need not be set, it is used as a convenient mechanism for accessing elements of COMM.

On intermediate re-entry: if **Pointers** = YES, MX need not be set.

If **Pointers** = NO, MX must contain the result of $y = Bx$ when IREVCM returns the value 2. It must return the imaginary parts of the computed shifts when IREVCM returns the value 3.

On intermediate exit: if **Pointers** = YES, MX is not referenced.

If **Pointers** = NO, MX contains the vector Bx when IREVCM returns the value $+1$.

On final exit: does not contain any useful data.

7: NSHIFT – INTEGER *Output*

On intermediate exit: if the option **Supplied Shifts** is set and IREVCM returns a value of 3, NSHIFT returns the number of complex shifts required.

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

Note: the dimension of the array COMM must be at least $\max(1, \text{LCOMM})$ (see F12AAF).

On initial entry: must remain unchanged following a call to the setup routine F12AAF.

On exit: contains data defining the current state of the iterative process.

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

Note: the dimension of the array ICOMM must be at least $\max(1, \text{LICOMM})$ (see F12AAF).

On initial entry: must remain unchanged following a call to the setup routine F12AAF.

On exit: contains data defining the current state of the iterative process.

10: IFAIL – INTEGER *Input/Output*

On initial 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, because for this routine the values of the output arguments may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On intermediate exit: the value of IFAIL is meaningless and should be ignored.

On final exit: (i.e., when IREVCM = 5) 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 initial entry, the maximum number of iterations ≤ 0, the option **Iteration Limit** has been set to a non-positive value.

IFAIL $= 2$

The options **Generalized** and **Regular** are incompatible.

IFAIL $= 3$

The option **Initial Residual** was selected but the starting vector held in RESID is zero.

IFAIL $= 4$

The maximum number of iterations has been reached. Some Ritz values may have converged; a subsequent call to F12ACF will return the number of converged values and the converged values.

IFAIL $= 5$

No shifts could be applied during a cycle of the implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV (see Section 5 in F12AAF for details of these arguments).

IFAIL $= 6$

Could not build an Arnoldi factorization. Consider changing NCV or NEV in the initialization routine (see Section 5 in F12AAF for details of these arguments).

IFAIL $= 7$

Unexpected error in internal call to compute eigenvalues and corresponding error bounds of the current upper Hessenberg matrix. Please contact NAG.

IFAIL $= 8$

Either the initialization routine F12AAF has not been called prior to the first call of this routine or a communication array has become corrupted.

IFAIL $= 9$

An unexpected error has occurred. Please contact NAG.

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

The relative accuracy of a Ritz value, λ, is considered acceptable if its Ritz estimate $\leq$ **Tolerance** $\times |\lambda|$. The default **Tolerance** used is the ***machine precision*** given by X02AJF.

8 Parallelism and Performance

F12ABF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

F12ABF 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

None.

10 Example

This example solves $Ax = \lambda x$ in shift-invert 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. The shift used is a real number.

10.1 Program Text

```
    Program f12abfe

!     F12ABF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: dgttrf, dgttrs, dnrm2, f12aaf, f12abf, f12acf,    &
                             f12adf, f12aef, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter    :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: two = 2.0_nag_wp
      Integer, Parameter               :: imon = 0, nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: h, rho, s, s1, s2, s3, sigmai,       &
                                          sigmar
      Integer                          :: i, ifail, ifail1, info, irevcm,      &
                                          lcomm, ldv, licomm, n, nconv, ncv,   &
                                          nev, niter, nshift, nx
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: comm(:), d(:,:), dd(:), dl(:),       &
                                          du(:), du2(:), mx(:), resid(:),      &
                                          v(:,:), x(:)
      Integer, Allocatable             :: icomm(:), ipiv(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: real
!     .. Executable Statements ..
      Write (nout,*) 'F12ABF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) nx, nev, ncv, rho, sigmar, sigmai
      n = nx*nx
      ldv = n
      licomm = 140
      lcomm = 3*n + 3*ncv*ncv + 6*ncv + 60
      Allocate (comm(lcomm),d(ncv,3),dd(n),dl(n),du(n),du2(n),mx(n),resid(n),  &
        v(ldv,ncv),x(n),icomm(licomm),ipiv(n))

!     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 mode.
      ifail = 0
      Call f12adf('SHIFTED INVERSE REAL',icomm,comm,ifail)

!     Construct C = A - SIGMA*I, and factorize using DGTTRF/F07CDF.
      h = one/real(n+1,kind=nag_wp)
      s = rho*h/two
      s1 = -one - s
      s2 = two - sigmar
      s3 = -one + s
      dl(1:n-1) = s1
      dd(1:n-1) = s2
      du(1:n-1) = s3
      dd(n) = s2

!     The NAG name equivalent of dgttrf is f07cdf
      Call dgttrf(n,dl,dd,du,du2,ipiv,info)

      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  x <--- OP*x = inv[A-SIGMA*I]*x.
!           The NAG name equivalent of dgttrs is f07cef
            Call dgttrs('N',n,1,dl,dd,du,du2,ipiv,x,n,info)
          Else If (irevcm==4 .And. imon/=0) Then
!           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/vectors.
        ifail1 = 0
        Call f12acf(nconv,d,d(1,2),v,ldv,sigmar,sigmai,resid,v,ldv,comm,icomm, &
          ifail1)
!       Print computed eigenvalues.
        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 closest to unity are:',/)
99997 Format (1X,I8,5X,'( ',F12.4,' , ',F12.4,' )')
    End Program f12abfe
```

10.2 Program Data

```
F12ABF Example Program Data
 10 4 20 10.0 1.0 0.0 : Values for NX NEV NCV RHO SIGMAR and SIGMAI
```

10.3 Program Results

```
 F12ABF Example Program Results


 The    4 Ritz values of closest to unity are:

        1     (       1.0192 ,       0.0000 )
        2     (       0.9656 ,       0.0000 )
        3     (       1.0738 ,       0.0000 )
        4     (       0.9129 ,       0.0000 )
```