NAG Library Routine Document

F12ACF

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

F12ACF is a post-processing routine that must be called following a final exit from F12ABF. These are part of a suite of routines for the solution of real sparse eigensystems. The suite also includes F12AAF, F12ADF and F12AEF.

2 Specification

```
SUBROUTINE F12ACF (NCONV, DR, DI, Z, LDZ, SIGMAR, SIGMAI, RESID, V, LDV,    &
                   COMM, ICOMM, IFAIL)
INTEGER            NCONV, LDZ, LDV, ICOMM(*), IFAIL
REAL (KIND=nag_wp) DR(*), DI(*), Z(LDZ,*), SIGMAR, SIGMAI, RESID(*),       &
                   V(LDV,*), 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.

Following a call to F12ABF, F12ACF returns the converged approximations to eigenvalues and (optionally) the corresponding approximate eigenvectors and/or an orthonormal basis for the associated approximate invariant subspace. The eigenvalues (and eigenvectors) are selected from those of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices. There is negligible additional cost to obtain eigenvectors; an orthonormal basis is always computed, but there is an additional storage cost if both are requested.

F12ACF is based on the routine **dneupd** from the ARPACK package, which uses 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 some of the interfaces.

F12ACF, is a post-processing routine that must be called following a successful final exit from F12ABF. F12ACF uses data returned from F12ABF and options, set either by default or explicitly by calling F12ADF, to return the converged approximations to selected eigenvalues and (optionally):

– the corresponding approximate eigenvectors;

– an orthonormal basis for the associated approximate invariant subspace;

– both.

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: NCONV – INTEGER *Output*

On exit: the number of converged eigenvalues as found by F12ABF.

2: DR(∗) – REAL (KIND=nag_wp) array *Output*

Note: the dimension of the array DR must be at least NEV (see F12AAF).

On exit: the first NCONV locations of the array DR contain the real parts of the converged approximate eigenvalues.

3: DI(∗) – REAL (KIND=nag_wp) array *Output*

Note: the dimension of the array DI must be at least NEV (see F12AAF).

On exit: the first NCONV locations of the array DI contain the imaginary parts of the converged approximate eigenvalues.

4: Z(LDZ, ∗) – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array Z must be at least NEV + 1 if the default option **Vectors** = RITZ has been selected and at least 1 if the option **Vectors** = NONE or SCHUR has been selected (see F12AAF).

On exit: if the default option **Vectors** = RITZ (see F12ADF) has been selected then Z contains the final set of eigenvectors corresponding to the eigenvalues held in DR and DI. The complex eigenvector associated with the eigenvalue with positive imaginary part is stored in two consecutive columns. The first column holds the real part of the eigenvector and the second column holds the imaginary part. The eigenvector associated with the eigenvalue with negative imaginary part is simply the complex conjugate of the eigenvector associated with the positive imaginary part.

5: LDZ – INTEGER *Input*

On entry: the first dimension of the array Z as declared in the (sub)program from which F12ACF is called.

Constraints:

if the default option **Vectors** = Ritz has been selected, LDZ ≥ N;
if the option **Vectors** = None or Schur has been selected, LDZ ≥ 1.

6: SIGMAR – REAL (KIND=nag_wp) *Input*

On entry: if one of the **Shifted Inverse Real** modes have been selected then SIGMAR contains the real part of the shift used; otherwise SIGMAR is not referenced.

7: SIGMAI – REAL (KIND=nag_wp) *Input*

On entry: if one of the **Shifted Inverse Real** modes have been selected then SIGMAI contains the imaginary part of the shift used; otherwise SIGMAI is not referenced.

8: RESID(∗) – REAL (KIND=nag_wp) array *Input*

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

On entry: must not be modified following a call to F12ABF since it contains data required by F12ACF.

9: 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 entry: the NCV columns of V contain the Arnoldi basis vectors for OP as constructed by F12ABF.

On exit: if the option **Vectors** = SCHUR has been set, or the option **Vectors** = RITZ has been set and a separate array Z has been passed (i.e., Z does not equal V), then the first NCONV columns of V will contain approximate Schur vectors that span the desired invariant subspace.

10: LDV – INTEGER *Input*

On entry: the first dimension of the array V as declared in the (sub)program from which F12ACF is called.

Constraint: LDV ≥ N.

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

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

On initial entry: must remain unchanged from the prior call to F12ABF.

On exit: contains data on the current state of the solution.

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

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

On initial entry: must remain unchanged from the prior call to F12ABF.

On exit: contains data on the current state of the solution.

13: 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, LDZ $<$ max$(1, \mathrm{N})$ or LDZ < 1 when no vectors are required.

IFAIL $= 2$

On entry, the option **Vectors** $=$ Select was selected, but this is not yet implemented.

IFAIL $= 3$

The number of eigenvalues found to sufficient accuracy prior to calling F12ACF, as communicated through the argument ICOMM, is zero.

IFAIL $= 4$

The number of converged eigenvalues as calculated by F12ABF differ from the value passed to it through the argument ICOMM.

IFAIL $= 5$

Unexpected error during calculation of a real Schur form: there was a failure to compute all the converged eigenvalues. Please contact NAG.

IFAIL $= 6$

Unexpected error: the computed Schur form could not be reordered by an internal call. Please contact NAG.

IFAIL $= 7$

Unexpected error in internal call while calculating eigenvectors. Please contact NAG.

IFAIL $= 8$

Either the solver routine F12ABF has not been called prior to the call of this routine or a communication array has become corrupted.

IFAIL $= 9$

The routine was unable to dynamically allocate sufficient internal workspace. Please contact NAG.

IFAIL $= 10$

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

F12ACF 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 $Ax = \lambda Bx$ in regular-invert mode, where A and B are obtained from the standard central difference discretization of the one-dimensional convection-diffusion operator $\frac{d^2u}{dx^2} + \rho\frac{du}{dx}$ on $[0, 1]$, with zero Dirichlet boundary conditions.

10.1 Program Text

```
!   F12ACF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module f12acfe_mod

!     F12ACF 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, mv
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: one = 1.0_nag_wp
      Integer, Parameter, Public         :: imon = 0, nin = 5, nout = 6
    Contains
      Subroutine av(nx,rho,v,w)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: two = 2.0_nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: rho
        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)             :: dd, dl, du, h, s
```

```
        Integer                       :: j, n
!       .. Intrinsic Procedures ..
        Intrinsic                     :: real
!       .. Executable Statements ..
        n = nx*nx
        h = one/real(n+1,kind=nag_wp)
        s = rho/two
        dd = two/h
        dl = -one/h - s
        du = -one/h + s
        w(1) = dd*v(1) + du*v(2)
        Do j = 2, n - 1
          w(j) = dl*v(j-1) + dd*v(j) + du*v(j+1)
        End Do
        w(n) = dl*v(n-1) + dd*v(n)
        Return
      End Subroutine av

      Subroutine mv(nx,v,w)

!       .. Use Statements ..
        Use nag_library, Only: dscal
!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: four = 4.0_nag_wp
!       .. 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)            :: h
        Integer                       :: j, n
!       .. Intrinsic Procedures ..
        Intrinsic                     :: real
!       .. Executable Statements ..
        n = nx*nx
        w(1) = four*v(1) + one*v(2)
        Do j = 2, n - 1
          w(j) = one*v(j-1) + four*v(j) + one*v(j+1)
        End Do
        w(n) = one*v(n-1) + four*v(n)
        h = one/real(n+1,kind=nag_wp)
!       The NAG name equivalent of dscal is f06edf
        Call dscal(n,h,w,1)
        Return
      End Subroutine mv
    End Module f12acfe_mod

    Program f12acfe

!     F12ACF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: dnrm2, dpttrf, dpttrs, f12aaf, f12abf, f12acf,    &
                             f12adf, f12aef, nag_wp
      Use f12acfe_mod, Only: av, imon, mv, nin, nout, one
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: h, rho, sigmai, sigmar
      Integer                         :: ifail, ifail1, info, irevcm, j,      &
                                         lcomm, ldv, licomm, n, nconv, ncv,   &
                                         nev, niter, nshift, nx
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: comm(:), d(:,:), md(:), me(:),       &
                                         mx(:), resid(:), v(:,:), x(:)
      Integer, Allocatable            :: icomm(:)
!     .. Intrinsic Procedures ..
      Intrinsic                       :: real
!     .. Executable Statements ..
      Write (nout,*) 'F12ACF Example Program Results'
```

```
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) nx, nev, ncv, rho
      n = nx*nx
      ldv = n
      licomm = 140
      lcomm = 3*n + 3*ncv*ncv + 6*ncv + 60
      Allocate (comm(lcomm),d(ncv,3),md(n),me(n-1),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 mode.
      ifail = 0
      Call f12adf('REGULAR INVERSE',icomm,comm,ifail)
!     Set problem type.
      Call f12adf('GENERALIZED',icomm,comm,ifail)
!     Use pointers to Workspace in calculating matrix vector
!     products rather than interfacing through the array X
      Call f12adf('POINTERS=YES',icomm,comm,ifail)

!     Construct M, and factorize using DPTTRF/F07JDF.
      h = one/real(n+1,kind=nag_wp)
      md(1:n-1) = 4.0_nag_wp*h
      me(1:n-1) = h
      md(n) = 4.0_nag_wp*h

!     The NAG name equivalent of dpttrf is f07jdf
      Call dpttrf(n,md,me,info)

      irevcm = 0

      ifail = -1
loop: Do
        Call f12abf(irevcm,resid,v,ldv,x,mx,nshift,comm,icomm,ifail)

        If (irevcm/=5) Then
          Select Case (irevcm)
          Case (-1,1)
!           Perform  y <--- OP*x = inv[M]*A*x using DPTTRS/F07JEF.
            Call av(nx,rho,comm(icomm(1)),comm(icomm(2)))
!           The NAG name equivalent of dpttrs is f07jef
            Call dpttrs(n,1,md,me,comm(icomm(2)),n,info)
          Case (2)
!           Perform  y <--- M*x.
            Call mv(nx,comm(icomm(1)),comm(icomm(2)))
          Case (4)
            If (imon/=0) Then
!             Output monitoring information if required.
              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
          End Select
        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 j = 1, nconv
          Write (nout,99997) j, d(j,1), d(j,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,' generalized Ritz values of largest ',          &
        'magnitude are:',/)
99997 Format (1X,I8,5X,'( ',F12.4,' , ',F12.4,' )')
    End Program f12acfe
```

10.2 Program Data

```
F12ACF Example Program Data
 10 4 20 10.0 : Values for NX NEV NCV RHO
```

10.3 Program Results

```
 F12ACF Example Program Results


 The    4 generalized Ritz values of largest magnitude are:

        1     (   20383.0384 ,       0.0000 )
        2     (   20338.7563 ,       0.0000 )
        3     (   20265.2844 ,       0.0000 )
        4     (   20163.1142 ,       0.0000 )
```