NAG Library Routine Document

F02FJF

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

F02FJF finds eigenvalues and eigenvectors of a real sparse symmetric or generalized symmetric eigenvalue problem.

2 Specification

```
SUBROUTINE F02FJF (N, M, K, NOITS, TOL, DOT, IMAGE, MONIT, NOVECS, X,      &
                   LDX, D, WORK, LWORK, RUSER, LRUSER, IUSER, LIUSER,      &
                   IFAIL)
INTEGER            N, M, K, NOITS, NOVECS, LDX, LWORK, LRUSER,             &
                   IUSER(LIUSER), LIUSER, IFAIL
REAL (KIND=nag_wp) TOL, DOT, X(LDX,K), D(K), WORK(LWORK),                  &
                   RUSER(LRUSER)
EXTERNAL           DOT, IMAGE, MONIT
```

3 Description

F02FJF finds the m eigenvalues of largest absolute value and the corresponding eigenvectors for the real eigenvalue problem

$$Cx = \lambda x \tag{1}$$

where C is an n by n matrix such that

$$BC = C^{\mathrm{T}}B \tag{2}$$

for a given positive definite matrix B. C is said to be B-symmetric. Different specifications of C allow for the solution of a variety of eigenvalue problems. For example, when

$$C = A \quad \text{and} \quad B = I \quad \text{where} \quad A = A^{\mathrm{T}}$$

the routine finds the m eigenvalues of largest absolute magnitude for the standard symmetric eigenvalue problem

$$Ax = \lambda x. \tag{3}$$

The routine is intended for the case where A is sparse.

As a second example, when

$$C = B^{-1}A$$

where

$$A = A^{\mathrm{T}}$$

the routine finds the m eigenvalues of largest absolute magnitude for the generalized symmetric eigenvalue problem

$$Ax = \lambda Bx. \tag{4}$$

The routine is intended for the case where A and B are sparse.

The routine does not require C explicitly, but C is specified via IMAGE which, given an n-element vector z, computes the image w given by

$$w = Cz.$$

For instance, in the above example, where $C = B^{-1}A$, IMAGE will need to solve the positive definite system of equations $Bw = Az$ for w.

To find the m eigenvalues of smallest absolute magnitude of (3) we can choose $C = A^{-1}$ and hence find the reciprocals of the required eigenvalues, so that IMAGE will need to solve $Aw = z$ for w, and correspondingly for (4) we can choose $C = A^{-1}B$ and solve $Aw = Bz$ for w.

A table of examples of choice of IMAGE is given in Table 1. It should be remembered that the routine also returns the corresponding eigenvectors and that B is positive definite. Throughout A is assumed to be symmetric and, where necessary, nonsingularity is also assumed.

Eigenvalues Required	**Problem**		
	$Ax = \lambda x (B = I)$	$Ax = \lambda Bx$	$ABx = \lambda x$
Largest	Compute $w = Az$	Solve $Bw = Az$	Compute $w = ABz$
Smallest (Find $1/\lambda$)	Solve $Aw = z$	Solve $Aw = Bz$	Solve $Av = z$, $Bw = v$
Furthest from σ (Find $\lambda - \sigma$)	Compute $w = (A - \sigma I)z$	Solve $Bw = (A - \sigma B)z$	Compute $w = (AB - \sigma I)z$
Closest to σ (Find $1/(\lambda - \sigma)$)	Solve $(A - \sigma I)w = z$	Solve $(A - \sigma B)w = Bz$	Solve $(AB - \sigma I)w = z$

Table 1
The Requirement of IMAGE for Various Problems.

The matrix B also need not be supplied explicitly, but is specified via DOT which, given n-element vectors z and w, computes the generalized dot product $w^{\mathrm{T}}Bz$.

F02FJF is based upon routine SIMITZ (see Nikolai (1979)), which is itself a derivative of the Algol procedure ritzit (see Rutishauser (1970)), and uses the method of simultaneous (subspace) iteration. (See Parlett (1998) for a description, analysis and advice on the use of the method.)

The routine performs simultaneous iteration on $k > m$ vectors. Initial estimates to $p \leq k$ eigenvectors, corresponding to the p eigenvalues of C of largest absolute value, may be supplied to F02FJF. When possible k should be chosen so that the kth eigenvalue is not too close to the m required eigenvalues, but if k is initially chosen too small then F02FJF may be re-entered, supplying approximations to the k eigenvectors found so far and with k then increased.

At each major iteration F02FJF solves an r by r ($r \leq k$) eigenvalue sub-problem in order to obtain an approximation to the eigenvalues for which convergence has not yet occurred. This approximation is refined by Chebyshev acceleration.

4 References

Nikolai P J (1979) Algorithm 538: Eigenvectors and eigenvalues of real generalized symmetric matrices by simultaneous iteration *ACM Trans. Math. Software* **5** 118–125

Parlett B N (1998) *The Symmetric Eigenvalue Problem* SIAM, Philadelphia

Rutishauser H (1969) Computational aspects of F L Bauer's simultaneous iteration method *Numer. Math.* **13** 4–13

Rutishauser H (1970) Simultaneous iteration method for symmetric matrices *Numer. Math.* **16** 205–223

5 Arguments

1: N – INTEGER *Input*

On entry: n, the order of the matrix C.

Constraint: N ≥ 1.

2: M – INTEGER *Input/Output*

On entry: m, the number of eigenvalues required.

Constraint: M ≥ 1.

On exit: m', the number of eigenvalues actually found. It is equal to m if IFAIL $= 0$ on exit, and is less than m if IFAIL $= 2$, 3 or 4. See Sections 6 and 9 for further information.

3: K – INTEGER *Input*

On entry: the number of simultaneous iteration vectors to be used. Too small a value of K may inhibit convergence, while a larger value of K incurs additional storage and additional work per iteration.

Suggested value: K = M + 4 will often be a reasonable choice in the absence of better information.

Constraint: M $<$ K $\leq$ N.

4: NOITS – INTEGER *Input/Output*

On entry: the maximum number of major iterations (eigenvalue sub-problems) to be performed. If NOITS ≤ 0, the value 100 is used in place of NOITS.

On exit: the number of iterations actually performed.

5: TOL – REAL (KIND=nag_wp) *Input*

On entry: a relative tolerance to be used in accepting eigenvalues and eigenvectors. If the eigenvalues are required to about t significant figures, TOL should be set to about 10^{-t}. d_i is accepted as an eigenvalue as soon as two successive approximations to d_i differ by less than $\left(\left|\tilde{d}_i\right| \times \text{TOL}\right)/10$, where $\tilde{d}_i$ is the latest approximation to d_i. Once an eigenvalue has been accepted, an eigenvector is accepted as soon as $(d_i f_i)/(d_i - d_k) <$ TOL, where f_i is the normalized residual of the current approximation to the eigenvector (see Section 9 for further information). The values of the f_i and d_i can be printed from MONIT. If TOL is supplied outside the range $(\epsilon, 1.0)$, where ϵ is the ***machine precision***, the value ϵ is used in place of TOL.

6: DOT – REAL (KIND=nag_wp) FUNCTION, supplied by the user. *External Procedure*

DOT must return the value $w^{\mathrm{T}}Bz$ for given vectors w and z. For the standard eigenvalue problem, where $B = I$, DOT must return the dot product $w^{\mathrm{T}}z$.

The specification of DOT is:

```
FUNCTION DOT (IFLAG, N, Z, W, RUSER, LRUSER, IUSER, LIUSER)
REAL (KIND=nag_wp) DOT
INTEGER            IFLAG, N, LRUSER, IUSER(LIUSER), LIUSER
REAL (KIND=nag_wp) Z(N), W(N), RUSER(LRUSER)
```

1: IFLAG – INTEGER *Input/Output*

On entry: is always non-negative.

On exit: may be used as a flag to indicate a failure in the computation of $w^{\mathrm{T}}Bz$. If IFLAG is negative on exit from DOT, F02FJF will exit immediately with IFAIL set to IFLAG. Note that in this case DOT must still be assigned a value.

2: N – INTEGER *Input*

On entry: the number of elements in the vectors z and w and the order of the matrix B.

3: Z(N) – REAL (KIND=nag_wp) array *Input*

On entry: the vector z for which $w^{\mathrm{T}}Bz$ is required.

4: W(N) – REAL (KIND=nag_wp) array *Input*

On entry: the vector w for which $w^{\mathrm{T}}Bz$ is required.

5: RUSER(LRUSER) – REAL (KIND=nag_wp) array *User Workspace*

DOT is called with the argument RUSER as supplied to F02FJF. You should use the array RUSER to supply information to DOT.

6: LRUSER – INTEGER *Input*

On entry: the dimension of the array RUSER as declared in the (sub)program from which F02FJF is called.

7: IUSER(LIUSER) – INTEGER array *User Workspace*

DOT is called with the argument IUSER as supplied to F02FJF. You should use the array IUSER to supply information to DOT.

8: LIUSER – INTEGER *Input*

On entry: the dimension of the array IUSER as declared in the (sub)program from which F02FJF is called.

DOT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which F02FJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

7: IMAGE – SUBROUTINE, supplied by the user. *External Procedure*

IMAGE must return the vector $w = Cz$ for a given vector z.

The specification of IMAGE is:

```
SUBROUTINE IMAGE (IFLAG, N, Z, W, RUSER, LRUSER, IUSER, LIUSER)
INTEGER            IFLAG, N, LRUSER, IUSER(LIUSER), LIUSER
REAL (KIND=nag_wp) Z(N), W(N), RUSER(LRUSER)
```

1: IFLAG – INTEGER *Input/Output*

On entry: is always non-negative.

On exit: may be used as a flag to indicate a failure in the computation of w. If IFLAG is negative on exit from IMAGE, F02FJF will exit immediately with IFAIL set to IFLAG.

2: N – INTEGER *Input*

On entry: n, the number of elements in the vectors w and z, and the order of the matrix C.

3: Z(N) – REAL (KIND=nag_wp) array *Input*

On entry: the vector z for which Cz is required.

4: W(N) – REAL (KIND=nag_wp) array *Output*

On exit: the vector $w = Cz$.

5: RUSER(LRUSER) – REAL (KIND=nag_wp) array *User Workspace*

IMAGE is called with the argument RUSER as supplied to F02FJF. You should use the array RUSER to supply information to IMAGE.

6: LRUSER – INTEGER *Input*

On entry: the dimension of the array RUSER as declared in the (sub)program from which F02FJF is called.

7: IUSER(LIUSER) – INTEGER array *User Workspace*

IMAGE is called with the argument IUSER as supplied to F02FJF. You should use the array IUSER to supply information to IMAGE.

8: LIUSER – INTEGER *Input*

On entry: the dimension of the array IUSER as declared in the (sub)program from which F02FJF is called.

IMAGE must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which F02FJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

8: MONIT – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

MONIT is used to monitor the progress of F02FJF. MONIT may be the dummy subroutine F02FJZ if no monitoring is actually required. (F02FJZ is included in the NAG Library.) MONIT is called after the solution of each eigenvalue sub-problem and also just prior to return from F02FJF. The arguments ISTATE and NEXTIT allow selective printing by MONIT.

The specification of MONIT is:

```
SUBROUTINE MONIT (ISTATE, NEXTIT, NEVALS, NEVECS, K, F, D)
INTEGER            ISTATE, NEXTIT, NEVALS, NEVECS, K
REAL (KIND=nag_wp) F(K), D(K)
```

1: ISTATE – INTEGER *Input*

On entry: specifies the state of F02FJF.

ISTATE = 0
: No eigenvalue or eigenvector has just been accepted.

ISTATE = 1
: One or more eigenvalues have been accepted since the last call to MONIT.

ISTATE = 2
: One or more eigenvectors have been accepted since the last call to MONIT.

ISTATE = 3
: One or more eigenvalues and eigenvectors have been accepted since the last call to MONIT.

ISTATE = 4
: Return from F02FJF is about to occur.

2: NEXTIT – INTEGER *Input*

On entry: the number of the next iteration.

3: NEVALS – INTEGER *Input*

On entry: the number of eigenvalues accepted so far.

4: NEVECS – INTEGER *Input*

On entry: the number of eigenvectors accepted so far.

5: K – INTEGER *Input*

On entry: k, the number of simultaneous iteration vectors.

6: F(K) – REAL (KIND=nag_wp) array *Input*

On entry: a vector of error quantities measuring the state of convergence of the simultaneous iteration vectors. See TOL and Section 9 for further details. Each element of F is initially set to the value 4.0 and an element remains at 4.0 until the corresponding vector is tested.

7: D(K) – REAL (KIND=nag_wp) array *Input*

On entry: D(i) contains the latest approximation to the absolute value of the ith eigenvalue of C.

MONIT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which F02FJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

9: NOVECS – INTEGER *Input*

On entry: the number of approximate vectors that are being supplied in X. If NOVECS is outside the range $(0, \mathrm{K})$, the value 0 is used in place of NOVECS.

10: X(LDX, K) – REAL (KIND=nag_wp) array *Input/Output*

On entry: if $0 <$ NOVECS $\leq$ K, the first NOVECS columns of X must contain approximations to the eigenvectors corresponding to the NOVECS eigenvalues of largest absolute value of C. Supplying approximate eigenvectors can be useful when reasonable approximations are known, or when F02FJF is being restarted with a larger value of K. Otherwise it is not necessary to supply approximate vectors, as simultaneous iteration vectors will be generated randomly by F02FJF.

On exit: if IFAIL $= 0$, 2, 3 or 4, the first m' columns contain the eigenvectors corresponding to the eigenvalues returned in the first m' elements of D; and the next $k - m' - 1$ columns contain approximations to the eigenvectors corresponding to the approximate eigenvalues returned in the next $k - m' - 1$ elements of D. Here m' is the value returned in M, the number of eigenvalues actually found. The kth column is used as workspace.

11: LDX – INTEGER *Input*

On entry: the first dimension of the array X as declared in the (sub)program from which F02FJF is called.

Constraint: LDX $\geq$ N.

12: D(K) – REAL (KIND=nag_wp) array *Output*

On exit: if IFAIL $= 0$, 2, 3 or 4, the first m' elements contain the first m' eigenvalues in decreasing order of magnitude; and the next $k - m' - 1$ elements contain approximations to the next $k - m' - 1$ eigenvalues. Here m' is the value returned in M, the number of eigenvalues actually found. D(k) contains the value e where $(-e, e)$ is the latest interval over which Chebyshev acceleration is performed.

13: WORK(LWORK) – REAL (KIND=nag_wp) array *Workspace*

14: LWORK – INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which F02FJF is called.

Constraint: LWORK $\geq 3 \times \mathrm{K} + \max(\mathrm{K} \times \mathrm{K}, 2 \times \mathrm{N})$.

15: RUSER(LRUSER) – REAL (KIND=nag_wp) array *User Workspace*

RUSER is not used by F02FJF, but is passed directly to DOT and IMAGE and should be used to pass information to these routines.

16: LRUSER – INTEGER *Input*

On entry: the dimension of the array RUSER as declared in the (sub)program from which F02FJF is called.

Constraint: LRUSER ≥ 1.

17: IUSER(LIUSER) – INTEGER array *User Workspace*

IUSER is not used by F02FJF, but is passed directly to DOT and IMAGE and should be used to pass information to these routines.

18: LIUSER – INTEGER *Input*

On entry: the dimension of the array IUSER as declared in the (sub)program from which F02FJF is called.

Constraint: LIUSER ≥ 1.

19: 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, 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 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 < 0

A negative value of IFAIL indicates an exit from F02FJF because you have set IFLAG negative in DOT or IMAGE. The value of IFAIL will be the same as your setting of IFLAG.

IFAIL $= 1$

On entry, N < 1,
or M < 1,
or M $\geq$ K,
or K $>$ N,
or LDX $<$ N,

or LWORK $< 3 \times \mathrm{K} + \max(\mathrm{K} \times \mathrm{K}, 2 \times \mathrm{N})$,
or LRUSER < 1,
or LIUSER < 1.

IFAIL $= 2$

Not all the requested eigenvalues and vectors have been obtained. Approximations to the rth eigenvalue are oscillating rapidly indicating that severe cancellation is occurring in the rth eigenvector and so M is returned as $(r-1)$. A restart with a larger value of K may permit convergence.

IFAIL $= 3$

Not all the requested eigenvalues and vectors have been obtained. The rate of convergence of the remaining eigenvectors suggests that more than NOITS iterations would be required and so the input value of M has been reduced. A restart with a larger value of K may permit convergence.

IFAIL $= 4$

Not all the requested eigenvalues and vectors have been obtained. NOITS iterations have been performed. A restart, possibly with a larger value of K, may permit convergence.

IFAIL $= 5$

This error is very unlikely to occur, but indicates that convergence of the eigenvalue sub-problem has not taken place. Restarting with a different set of approximate vectors may allow convergence. If this error occurs you should check carefully that F02FJF is being called correctly.

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

Eigenvalues and eigenvectors will normally be computed to the accuracy requested by the argument TOL, but eigenvectors corresponding to small or to close eigenvalues may not always be computed to the accuracy requested by the argument TOL. Use of the MONIT to monitor acceptance of eigenvalues and eigenvectors is recommended.

8 Parallelism and Performance

F02FJF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

F02FJF 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

The time taken by F02FJF will be principally determined by the time taken to solve the eigenvalue sub-problem and the time taken by DOT and IMAGE. The time taken to solve an eigenvalue sub-problem is approximately proportional to nk^2. It is important to be aware that several calls to DOT and IMAGE may occur on each major iteration.

As can be seen from Table 1, many applications of F02FJF will require the IMAGE to solve a system of linear equations. For example, to find the smallest eigenvalues of $Ax = \lambda Bx$, IMAGE needs to solve equations of the form $Aw = Bz$ for w and routines from Chapters F01 and F04 will frequently be useful in this context. In particular, if A is a positive definite variable band matrix, F04MCF may be used after A has been factorized by F01MCF. Thus factorization need be performed only once prior to calling F02FJF. An illustration of this type of use is given in the example program.

An approximation $\tilde{d}_h$, to the ith eigenvalue, is accepted as soon as $\tilde{d}_h$ and the previous approximation differ by less than $\left|\tilde{d}_h\right| \times \text{TOL}/10$. Eigenvectors are accepted in groups corresponding to clusters of eigenvalues that are equal, or nearly equal, in absolute value and that have already been accepted. If d_r is the last eigenvalue in such a group and we define the residual r_j as

$$r_j = Cx_j - y_r$$

where y_r is the projection of Cx_j, with respect to B, onto the space spanned by $x_1, x_2, \ldots, x_r$, and x_j is the current approximation to the jth eigenvector, then the value f_i returned in MONIT is given by

$$f_i = \max \left\|r_j\right\|_B / \left\|Cx_j\right\|_B \quad \|x\|_B^2 = x^{\mathrm{T}} Bx$$

and each vector in the group is accepted as an eigenvector if

$$(|d_r| f_r)/(|d_r| - e) < \text{TOL},$$

where e is the current approximation to $\left|\tilde{d}_k\right|$. The values of the f_i are systematically increased if the convergence criteria appear to be too strict. See Rutishauser (1970) for further details.

The algorithm implemented by F02FJF differs slightly from SIMITZ (see Nikolai (1979)) in that the eigenvalue sub-problem is solved using the singular value decomposition of the upper triangular matrix R of the Gram–Schmidt factorization of Cx_r, rather than forming $R^{\mathrm{T}}R$.

10 Example

This example finds the four eigenvalues of smallest absolute value and corresponding eigenvectors for the generalized symmetric eigenvalue problem $Ax = \lambda Bx$, where A and B are the 16 by 16 matrices

$$A = -\frac{1}{4}\begin{pmatrix}
-4 & 1 & & & 1 & & & & & & & & & & & \\
1 & -4 & 1 & & & 1 & & & & & & & & & & \\
 & 1 & -4 & 1 & & & 1 & & & & & & & & & \\
 & & 1 & -4 & 1 & & & 1 & & & & & & & & \\
1 & & & 1 & -4 & 1 & & & 1 & & & & & & & \\
 & 1 & & & 1 & -4 & 1 & & & 1 & & & & & & \\
 & & 1 & & & 1 & -4 & 1 & & & 1 & & & & & \\
 & & & 1 & & & 1 & -4 & 1 & & & 1 & & & & \\
 & & & & 1 & & & 1 & -4 & 1 & & & 1 & & & \\
 & & & & & 1 & & & 1 & -4 & 1 & & & 1 & & \\
 & & & & & & 1 & & & 1 & -4 & 1 & & & 1 & \\
 & & & & & & & 1 & & & 1 & -4 & 1 & & & \\
 & & & & & & & & 1 & & & 1 & -4 & 1 & & \\
 & & & & & & & & & 1 & & & 1 & -4 & 1 & \\
 & & & & & & & & & & 1 & & & 1 & -4 &
\end{pmatrix}$$

$$B = -\frac{1}{2}\begin{pmatrix}
-2 & 1 & & & & & & & & & & & & & & \\
1 & -2 & 1 & & & & & & & & & & & & & \\
 & 1 & -2 & 1 & & & & & & & & & & & & \\
 & & 1 & -2 & 1 & & & & & & & & & & & \\
 & & & 1 & -2 & 1 & & & & & & & & & & \\
 & & & & 1 & -2 & 1 & & & & & & & & & \\
 & & & & & 1 & -2 & 1 & & & & & & & & \\
 & & & & & & 1 & -2 & 1 & & & & & & & \\
 & & & & & & & 1 & -2 & 1 & & & & & & \\
 & & & & & & & & 1 & -2 & 1 & & & & & \\
 & & & & & & & & & 1 & -2 & 1 & & & & \\
 & & & & & & & & & & 1 & -2 & 1 & & & \\
 & & & & & & & & & & & 1 & -2 & 1 & & \\
 & & & & & & & & & & & & 1 & -2 & 1 & \\
 & & & & & & & & & & & & & 1 & -2 &
\end{pmatrix}$$

TOL is taken as 0.0001 and 6 iteration vectors are used. F11JAF is used to factorize the matrix A, prior to calling F02FJF, and F11JCF is used within IMAGE to solve the equations $Aw = Bz$ for w.

Output from MONIT occurs each time ISTATE is nonzero. Note that the required eigenvalues are the reciprocals of the eigenvalues returned by F02FJF.

10.1 Program Text

```
!   F02FJF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module f02fjfe_mod

!     F02FJF 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                             :: dot, image, monit
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter      :: half = 0.5_nag_wp
```

```
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Integer, Parameter, Public       :: nin = 5, nout = 6
    Contains
      Function dot(iflag,n,z,w,ruser,lruser,iuser,liuser)
!       This function implements the dot product - transpose(W)*B*Z.
!       DOT assumes that N is at least 3.

!       .. Function Return Value ..
        Real (Kind=nag_wp)             :: dot
!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: iflag
        Integer, Intent (In)           :: liuser, lruser, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: ruser(lruser)
        Real (Kind=nag_wp), Intent (In) :: w(n), z(n)
        Integer, Intent (Inout)        :: iuser(liuser)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: s
        Integer                        :: i
!       .. Executable Statements ..
        s = zero
        s = s + (z(1)-half*z(2))*w(1)
        s = s + (-half*z(n-1)+z(n))*w(n)
        Do i = 2, n - 1
          s = s + (-half*z(i-1)+z(i)-half*z(i+1))*w(i)
        End Do
        dot = s
!       Set iflag negative to terminate execution for any reason.
        iflag = 0
        Return
      End Function dot
      Subroutine image(iflag,n,z,w,ruser,lruser,iuser,liuser)
!       This routine solves  A*W = B*Z  for W.
!       The routine assumes that N is at least 3.

!       The data A, NNZ, LA, IROW, ICOL, IPIV and ISTR on exit from
!       F11JAF have been packed into the xUSER communication arrays in
!       the following way:
!       IUSER(1:2) = (/NNZ, LA/)
!       RUSER(1:LA) = A
!       IUSER(3:(2*LA+2*N+3)) = (/IROW, ICOL, IPIV, ISTR/)
!       We'll also use RUSER((LA+1):(LA+N)) as space for F11JCF's dummy
!       arg. B, and RUSER((LA+N+1):(LA+7*N+120)) as space for F11JCF's
!       dummy arg. WORK

!       .. Use Statements ..
        Use nag_library, Only: f11jcf, x02ajf
!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: iflag
        Integer, Intent (In)           :: liuser, lruser, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: ruser(lruser)
        Real (Kind=nag_wp), Intent (Out) :: w(n)
        Real (Kind=nag_wp), Intent (In) :: z(n)
        Integer, Intent (Inout)        :: iuser(liuser)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: rnorm, tol
        Integer                        :: ifail, itn, j, la, lwork, maxitn,    &
                                          nnz
        Character (2)                  :: method
!       .. Executable Statements ..
        nnz = iuser(1)
        la = iuser(2)

!       Form B*Z in RUSER((LA+1):(LA+N)) and initialize W to
!       zero.
        w(1:n) = zero
        ruser(la+1) = z(1) - half*z(2)
        Do j = 2, n - 1
          ruser(la+j) = -half*z(j-1) + z(j) - half*z(j+1)
        End Do
```

```
        ruser(la+n) = -half*z(n-1) + z(n)

!       Call F11JCF to solve the equations  A*W = B*Z.
        method = 'CG'
        tol = x02ajf()
        maxitn = 100
        lwork = 6*n + 120

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 1
        Call f11jcf(method,n,nnz,ruser,la,iuser(3),iuser(la+3),iuser(2*la+3),  &
          iuser(2*la+n+3),ruser(la+1),tol,maxitn,w,rnorm,itn,ruser(la+n+1),    &
          lwork,ifail)

        If (ifail>0) Then
          iflag = -ifail
        End If
        Return
      End Subroutine image
      Subroutine monit(istate,nextit,nevals,nevecs,k,f,d)
!       Monitoring routine for F02FJF.

!       .. Parameters ..
        Integer, Parameter             :: nout = 6
!       .. Scalar Arguments ..
        Integer, Intent (In)           :: istate, k, nevals, nevecs, nextit
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: d(k), f(k)
!       .. Local Scalars ..
        Integer                        :: i
!       .. Executable Statements ..
        If (istate/=0) Then
          Write (nout,*)
          Write (nout,99999) '  ISTATE = ', istate, ' NEXTIT = ', nextit
          Write (nout,99999) '  NEVALS = ', nevals, ' NEVECS = ', nevecs
          Write (nout,*) '       F           D'
          Write (nout,99998)(f(i),d(i),i=1,k)
        End If
        Return

99999   Format (1X,A,I4,A,I4)
99998   Format (1X,1P,E11.3,3X,E11.3)
      End Subroutine monit
    End Module f02fjfe_mod
    Program f02fjfe

!     F02FJF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: f02fjf, f06fef, f11jaf, nag_wp, x04cbf
      Use f02fjfe_mod, Only: dot, image, monit, nin, nout, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: dscale, dtol, tol
      Integer                          :: i, ifail, k, l, la, ldx, lfill,    &
                                          liuser, lruser, lwork, m, n, nnz,  &
                                          nnzc, noits, novecs, npivm
      Character (1)                    :: mic, pstrat
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:), d(:), ruser(:), work(:),     &
                                          x(:,:)
      Integer, Allocatable             :: icol(:), ipiv(:), irow(:), istr(:), &
                                          iuser(:)
      Character (1)                    :: clabs(1), rlabs(1)
!     .. Executable Statements ..
      Write (nout,*) 'F02FJF Example Program Results'
      Write (nout,*)
      Flush (nout)
!     Skip heading in data file
```

```
      Read (nin,*)
      Read (nin,*) n, m, k, tol
      la = 10*n
      ldx = n
      liuser = 2*la + 2*n + 3
      lruser = la + 7*n + 120
      lwork = 5*k + 2*n
      Allocate (a(la),d(n),ruser(lruser),work(lwork),x(ldx,k),icol(la),        &
        ipiv(n),irow(la),istr(n+1),iuser(liuser))

!     Set up the sparse symmetric coefficient matrix A.
      l = 0
      Do i = 1, n
        If (i>=5) Then
          l = l + 1
          a(l) = -0.25_nag_wp
          irow(l) = i
          icol(l) = i - 4
        End If
        If (i>=2) Then
          l = l + 1
          a(l) = -0.25_nag_wp
          irow(l) = i
          icol(l) = i - 1
        End If
        l = l + 1
        a(l) = 1.0_nag_wp
        irow(l) = i
        icol(l) = i
      End Do
      nnz = l

!     Call F11JAF to find an incomplete Cholesky factorization of A.
      lfill = 2
      dtol = zero
      mic = 'M'
      dscale = zero
      pstrat = 'M'

!     ifail: behaviour on error exit
!             =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call f11jaf(n,nnz,a,la,irow,icol,lfill,dtol,mic,dscale,pstrat,ipiv,istr, &
        nnzc,npivm,iuser,liuser,ifail)

!     Call F02FJF to find eigenvalues and eigenvectors.

      noits = 1000
      novecs = 0

!     Communicate A, NNZ, LA, IROW, ICOL, IPIV and ISTR to IMAGE
!     thread-safely using RUSER and IUSER.
!     In addition to using RUSER for storing A, we'll also use
!     7*N+120 elements of RUSER in place of local arrays in IMAGE.

!     Initialized A goes into ruser.
      ruser(1:nnz+nnzc) = a(1:nnz+nnzc)

!     NNZ, LA, IROW, ICOL, IPIV and ISTR go into IUSER, in that order.
!     Only the first NNZ+NNZC elements of IROW and ICOL will have been
!     initialized:
      iuser(1) = nnz
      iuser(2) = la
      iuser(3:2+nnz+nnzc) = irow(1:nnz+nnzc)
      iuser(la+3:la+2+nnz+nnzc) = icol(1:nnz+nnzc)
      iuser(2*la+3:2*la+2+n) = ipiv(1:n)
      iuser(liuser-n:liuser) = istr(1:n+1)

      ifail = -1
      Call f02fjf(n,m,k,noits,tol,dot,image,monit,novecs,x,ldx,d,work,lwork,   &
        ruser,lruser,iuser,liuser,ifail)
```

```
      If (ifail>=0) Then
        If (ifail/=1 .And. ifail<=4 .And. m>=1) Then
          Do i = 1, m
            d(i) = 1.0_nag_wp/d(i)
          End Do
          Write (nout,*) 'Final results'
          Write (nout,*)
          Write (nout,*) '  Eigenvalues'
          Write (nout,99999) d(1:m)
          Write (nout,*)
          Flush (nout)
!         Normalize eigenvectors
          Do i = 1, m
            Call f06fef(n,x(1,i),x(1,i),1)
          End Do
          Call x04cbf('General',' ',n,m,x,ldx,'1P,E12.3','  Eigenvectors','N', &
            rlabs,'N',clabs,80,0,ifail)
        End If
      End If

99999 Format (1X,1P,4E12.3)
    End Program f02fjfe
```

10.2 Program Data

```
F02FJF Example Program Data
 16 4 6 0.0001         : n, m, k, tol
```

10.3 Program Results

```
 F02FJF Example Program Results


   ISTATE =    3 NEXTIT =   17
   NEVALS =    1 NEVECS =    1
        F           D
   1.246E-07     1.822E+00
   4.000E+00     1.695E+00
   4.000E+00     1.668E+00
   4.000E+00     1.460E+00
   4.000E+00     1.275E+00
   4.000E+00     1.132E+00

   ISTATE =    4 NEXTIT =   30
   NEVALS =    4 NEVECS =    4
        F           D
   1.246E-07     1.822E+00
   2.450E-09     1.695E+00
   7.922E-09     1.668E+00
   3.210E-07     1.460E+00
   4.000E+00     1.275E+00
   4.000E+00     1.153E+00
 Final results

   Eigenvalues
    5.488E-01   5.900E-01   5.994E-01   6.850E-01

   Eigenvectors
     1.000E+00   1.000E+00   1.000E+00   1.000E+00
    -1.159E+00  -8.089E-01   1.127E+00  -1.237E+00
     1.168E+00  -7.555E-01  -1.070E+00   1.925E+00
    -1.130E+00   7.444E-01  -1.351E+00  -1.318E+00
     1.692E+00   1.494E+00   1.827E+00   8.027E-01
    -1.880E+00  -1.283E+00   1.793E+00  -4.766E-01
     1.885E+00  -1.251E+00  -1.759E+00   1.481E+00
    -1.760E+00   1.354E+00  -2.015E+00  -6.525E-01
     1.760E+00   1.354E+00   2.015E+00  -6.525E-01
    -1.885E+00  -1.251E+00   1.759E+00   1.481E+00
     1.880E+00  -1.283E+00  -1.793E+00  -4.766E-01
```

```
 -1.692E+00   1.494E+00  -1.827E+00   8.027E-01
  1.130E+00   7.444E-01   1.351E+00  -1.318E+00
 -1.168E+00  -7.555E-01   1.070E+00   1.925E+00
  1.159E+00  -8.089E-01  -1.127E+00  -1.237E+00
 -1.000E+00   1.000E+00  -1.000E+00   1.000E+00
```