NAG Toolbox for MATLAB

f02fj

1 Purpose

f02fj finds eigenvalues and eigenvectors of a real sparse symmetric or generalized symmetric eigenvalue problem.

2 Syntax

```
[m, noits, x, d, user, ifail] = f02fj(n, m, noits, tol, dot, image,
monit, novecs, x, lruser, liuser, 'k', k, 'user', user)
```

3 Description

f02fj 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

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

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

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

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

As a second example, when

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

where

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

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

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

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

The function does not require C explicitly, but C is specified via a user-supplied (sub)program **image** which, given an n element vector z, computes the image w given by

$$w = Cz.$$

For instance, in the above example, where $C = B^{-1}A$, the **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 user-supplied (sub)program **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 user-supplied (sub)program **image** is given in Table 1. It should be remembered that the function 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 User-supplied (Sub)program **image** for Various Problems.

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

f02fj 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 function 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 by you to f02fj. 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 f02fj may be re-entered, supplying approximations to the k eigenvectors found so far and with k then increased.

At each major iteration f02fj 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 Parameters

5.1 Compulsory Input Parameters

1: **n – int32 scalar**

n, the order of the matrix C.

Constraint: **n** ≥ 1.

2: **m – int32 scalar**

m, the number of eigenvalues required.

Constraint: **m** ≥ 1.

3: **noits – int32 scalar**

The maximum number of major iterations (eigenvalue sub-problems) to be performed. If **noits** ≤ 0, the value 100 is used in place of **noits**.

4: **tol – double scalar**

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 $\left(\left|\tilde{d}_i\right| \times \textbf{tol}\right)/10$, where $\tilde{d}_i$ is the latest approximation to d_i. Once an eigenvalue has been accepted, an eigenvector is accepted as soon as $(d_i f_i)/(d_i - d_k) < \textbf{tol}$, where f_i is the normalized residual of the current approximation to the eigenvector (see Section 8 for further information). The values of the f_i and d_i can be printed from the (sub)program **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**.

5: **dot – string containing name of m-file**

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 $w^{\mathrm{T}}z$.

dot is called from f02fj with the parameters **user**, **lruser**, **user** and **liuser** as supplied to f02fj. You are free to use the arrays **user** and **user** to supply information to **dot** and to user-supplied (sub)program **image** as an alternative to using global variables.

Its specification is:

```
[result, iflag, user] = dot(iflag, n, z, w, user, lruser, liuser)
```

Input Parameters

1: **iflag – int32 scalar**

Is always nonnegative.

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**, f02fj will exit immediately with **ifail** set to **iflag**. Note that in this case **dot** must still be assigned a value.

2: **n – int32 scalar**

The number of elements in the vectors z and w and the order of the matrix B.

3: **z(n) – double array**

The vector z for which $w^{\mathrm{T}}Bz$ is required.

4: **w(n) – double array**

The vector w for which $w^{\mathrm{T}}Bz$ is required.

5: **user – Any MATLAB object**
6: **lruser – int32 scalar**
7: **liuser – int32 scalar**

Output Parameters

1: **result – double scalar**

The result of the function.

2: **iflag – int32 scalar**

Is always nonnegative.

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**, f02fj will exit immediately with **ifail** set to **iflag**. Note that in this case **dot** must still be assigned a value.

3: **user – Any MATLAB object**

6: **image – string containing name of m-file**

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

image is called from f02fj with the parameters **user**, **lruser**, **user** and **liuser** as supplied to f02fj. You are free to use the arrays **user** and **user** to supply information to **image** and user-supplied real function **dot** as an alternative to using global variables.

Its specification is:

```
[iflag, w, user] = image(iflag, n, z, user, lruser, liuser)
```

Input Parameters

1: **iflag – int32 scalar**

Is always nonnegative.

May be used as a flag to indicate a failure in the computation of w. If **iflag** is negative on exit from **image**, f02fj will exit immediately with **ifail** set to **iflag**.

2: **n – int32 scalar**

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

3: **z(n) – double array**

The vector z for which Cz is required.

4: **user – Any MATLAB object**
5: **lruser – int32 scalar**
6: **liuser – int32 scalar**

Output Parameters

1: **iflag – int32 scalar**

Is always nonnegative.

May be used as a flag to indicate a failure in the computation of w. If **iflag** is negative on exit from **image**, f02fj will exit immediately with **ifail** set to **iflag**.

2: **w(n) – double array**

The vector $w = Cz$.

3: **user – Any MATLAB object**

7: **monit – string containing name of m-file**

monit is used to monitor the progress of f02fj. **monit** may be the dummy (sub)program **f02fjz** if no monitoring is actually required. **f02fjz** is included in the NAG Fortran Library. **monit** is called after the solution of each eigenvalue sub-problem and also just prior to return from f02fj. The parameters **istate** and **nextit** allow selective printing by **monit**.

Its specification is:

```
[] = monit(istate, nextit, nevals, nevecs, k, f, d)
```

Input Parameters

1: **istate – int32 scalar**

Specifies the state of f02fj and will have values as follows:

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 f02fj is about to occur.

2: **nextit – int32 scalar**

The number of the next iteration.

3: **nevals – int32 scalar**

The number of eigenvalues accepted so far.

4: **nevecs – int32 scalar**

The number of eigenvectors accepted so far.

5: **k – int32 scalar**

k, the number of simultaneous iteration vectors.

6: **f(k) – double array**

A vector of error quantities measuring the state of convergence of the simultaneous iteration vectors. See the parameter **tol** of f02fj above and Section 8 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) – double array**

d(i) contains the latest approximation to the absolute value of the ith eigenvalue of C.

Output Parameters

8: **novecs – int32 scalar**

The number of approximate vectors that are being supplied in **x**. If **novecs** is outside the range $(0, \mathbf{k})$, the value 0 is used in place of **novecs**.

9: **x(ldx,k) – double array**

ldx, the first dimension of the array, must be at least **n**.

If $0 < \mathbf{novecs} \leq \mathbf{k}$, 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 the function 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 the function.

10: **lruser – int32 scalar**

Constraint: **lruser** ≥ 1.

11: **liuser – int32 scalar**

Constraint: **liuser** ≥ 1.

5.2 Optional Input Parameters

1: **k – int32 scalar**

Default: The dimension of the arrays **d**, **x**. (An error is raised if these dimensions are not equal.)

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: $\mathbf{k} = \mathbf{m} + 4$ will often be a reasonable choice in the absence of better information.

Default: $\mathbf{m} + 4$

Constraint: $\mathbf{m} < \mathbf{k} \leq \mathbf{n}$.

2: **user – Any MATLAB object**

user is not used by f02fj, but is passed to **dot**, **image** and **monit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

ldx, work, lwork

5.4 Output Parameters

1: **m – int32 scalar**

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 8 for further information.

2: **noits – int32 scalar**

The number of iterations actually performed.

3: **x(ldx,k) – double array**

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.

4: **d(k) – double array**

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. $\mathbf{d}(k)$ contains the value e where $(-e, e)$ is the latest interval over which Chebyshev acceleration is performed.

5: **user – Any MATLAB object**

user is not used by f02fj, but is passed to **dot**, **image** and **monit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

6: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail < 0

A negative value of **ifail** indicates an exit from f02fj because you have set **iflag** negative in the user-supplied real function **dot** or user-supplied (sub)program **image**. The value of **ifail** will be the same as your setting of **iflag**.

ifail $= 1$

On entry, $\mathbf{n} < 1$,
or $\mathbf{m} < 1$,
or $\mathbf{m} \geq \mathbf{k}$,
or $\mathbf{k} > \mathbf{n}$,
or $\mathbf{ldx} < \mathbf{n}$,
or $\mathbf{lwork} < 3 \times \mathbf{k} + \max(\mathbf{k} \times \mathbf{k}, 2 \times \mathbf{n})$,
or $\mathbf{lruser} < 1$,
or $\mathbf{liuser} < 1$.

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 f02fj is being called correctly.

7 Accuracy

Eigenvalues and eigenvectors will normally be computed to the accuracy requested by the parameter **tol**, but eigenvectors corresponding to small or to close eigenvalues may not always be computed to the accuracy requested by the parameter **tol**. Use of the (sub)program **monit** to monitor acceptance of eigenvalues and eigenvectors is recommended.

8 Further Comments

The time taken by f02fj will be principally determined by the time taken to solve the eigenvalue sub-problem and the time taken by the user-supplied (sub)programs **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 f02fj will require the user-supplied (sub)program **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 functions from Chapters F01 and F04 will frequently be useful in this context. In particular, if A is a positive-definite variable band matrix, f04mc may be used after A has been factorized by f01mc. Thus factorization need be performed only once prior to calling f02fj. 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 $\left|\tilde{d}_h\right| \times \mathbf{tol}/10$. 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 (sub)program **monit** is given by

$$f_i = \max_{B} \left\|r_j\right\|_B / \left\|Cx_j\right\|_B \qquad \|x\|_B{}^2 = x^{\mathrm{T}}Bx$$

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

$$(|d_r|f_r)/(|d_r| - e) < \mathbf{tol},$$

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 f02fj 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 $R^{\mathrm{T}}R$.

9 Example

f02fj_dot.m

```
function [result, iflag, user] = f02fj_dot(iflag, n, z, w, user)
```

```
  b = user{2};
  result=transpose(w)*b*z;
```

f02fj_image.m

```
function [iflag, w, user] = f02fj_image(iflag, n, z, user)

  a=user{1};
  b=user{2};

  w=inv(a)*b*z;
```

f02fj_monit.m

```
function monit(istate, nextit, nevals, nevecs, k, f, d)

  if (istate ~= 0)
    fprintf('\n  istate = %d nextit = %d\n', istate, nextit);
    fprintf('  nevals = %d nevecs = %d\n', nevals, nevecs);
    fprintf('       f           d \n');
    for i=1:k
      fprintf('%11.3f %11.3f\n',f(i), d(i));
    end
  end
```

```
n = int32(16);
m = int32(4);
noits = int32(1000);
tol = 0.0001;
novecs = int32(0);
x = zeros(n, m+2);
% a, b will be passed in user
a  =  diag(ones(n,1))+diag(-0.25*ones(n-1,1),1)+diag(-0.25*ones(n-1,  1),-
1)+diag(-0.25*ones(n-4,1),4)+diag(-0.25*ones(n-4,1),-4);
b = diag(ones(n,1))+diag(-0.5*ones(n-1,1),1)+diag(-0.5*ones(n-1,1), -1);
[mOut, noitsOut, xOut, d, user, ifail] = ...
      f02fj(n, m, noits, tol, 'f02fj_dot', 'f02fj_image', 'f02fj_monit',
novecs, x, 'user', {a, b})
```

```
istate = 4 nextit = 30
  nevals = 4 nevecs = 4
       f           d
     0.000       1.822
     0.000       1.695
     0.000       1.668
     0.000       1.460
     4.000       1.275
     4.000       1.151
mOut =
           4
noitsOut =
          30
xOut =
    0.1189   -0.2153    0.1648   -0.1561   -0.4140   -0.5278
   -0.1378    0.1741    0.1858    0.1931    0.2343    0.2986
    0.1389    0.1626   -0.1763   -0.3005    0.1771    0.2258
   -0.1343   -0.1602   -0.2227    0.2058   -0.1627   -0.2074
    0.2012   -0.3217    0.3010   -0.1253   -0.2237   -0.2852
   -0.2235    0.2761    0.2954    0.0744    0.0671    0.0856
    0.2242    0.2692   -0.2899   -0.2312    0.1935    0.2467
   -0.2093   -0.2914   -0.3320    0.1018   -0.1668   -0.2126
    0.2093   -0.2914    0.3320    0.1018    0.1667    0.2126
   -0.2242    0.2692    0.2899   -0.2312   -0.1935   -0.2467
    0.2235    0.2761   -0.2954    0.0744   -0.0670   -0.0854
   -0.2012   -0.3217   -0.3010   -0.1253    0.2239    0.2854
```

```
    0.1343   -0.1602    0.2227    0.2058    0.1625    0.2072
   -0.1389    0.1626    0.1763   -0.3005   -0.1772   -0.2259
    0.1378    0.1741   -0.1858    0.1931   -0.2340   -0.2983
   -0.1189   -0.2153   -0.1648   -0.1561    0.4141    0.5279
d =
    1.8223
    1.6949
    1.6684
    1.4600
    1.2748
    1.1508
user =
    [16x16 double]    [16x16 double]
ifail =
           0
```