NAG Toolbox for MATLAB

f12aa

1 Purpose

f12aa is a setup function in a suite of functions consisting of f12aa, f12ab, f12ac, f12ad and f12ae. It is used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices.

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

2 Syntax

```
[icomm, comm, ifail] = f12aa(n, nev, ncv)
```

3 Description

The suite of functions 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.

f12aa is a setup function which must be called before f12ab, the reverse communication iterative solver, and before f12ad, the options setting function. f12ac is a post-processing function that must be called following a successful final exit from f12ab, while f12ae can be used to return additional monitoring information during the computation.

This setup function initializes the communication arrays, sets (to their default values) all options that can be set by you via the option setting function f12ad, and checks that the lengths of the communication arrays as passed by you are of sufficient length. For details of the options available and how to set them see Section 10.2 of the document for f12ad.

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 Parameters

5.1 Compulsory Input Parameters

1: **n – int32 scalar**

The order of the matrix A (and the order of the matrix B for the generalized problem) that defines the eigenvalue problem.

Constraint: $\mathbf{n} > 0$.

2: **nev – int32 scalar**

The number of eigenvalues to be computed.

Constraint: $0 < \mathbf{nev} < \mathbf{n} - 1$.

3: **ncv – int32 scalar**

The number of Arnoldi basis vectors to use during the computation.

At present there is no *a priori* analysis to guide the selection of **ncv** relative to **nev**. However, it is recommended that $\mathbf{ncv} \geq 2 \times \mathbf{nev} + 1$. If many problems of the same type are to be solved, you should experiment with varying **ncv** while keeping **nev** fixed for a given test problem. This will usually decrease the required number of matrix-vector operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal 'cross-over' with respect to CPU time is problem dependent and must be determined empirically.

Constraint: $\mathbf{nev} + 1 < \mathbf{ncv} \leq \mathbf{n}$.

5.2 Optional Input Parameters

None.

5.3 Input Parameters Omitted from the MATLAB Interface

licomm, lcomm

5.4 Output Parameters

1: **icomm**$(*)$ **– int32 array**

Note: the dimension of the array **icomm** must be at least $\max(1, \mathbf{licomm})$.

Contains data to be communicated to the other functions in the suite.

2: **comm**$(*)$ **– double array**

Note: the dimension of the array **comm** must be at least $\max(1, \mathbf{lcomm})$.

Contains data to be communicated to the other functions in the suite.

3: **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 $= 1$

On entry, $\mathbf{n} \leq 0$.

ifail $= 2$

On entry, $\mathbf{nev} \leq 0$.

ifail $= 3$

On entry, $\mathbf{ncv} < \mathbf{nev} + 2$ or $\mathbf{ncv} > \mathbf{n}$.

ifail $= 4$

On entry, $\mathbf{licomm} < 140$ and $\mathbf{licomm} \neq -1$.

ifail $= 5$

On entry, **lcomm** $< 3 \times$ **n** $+ 3 \times$ **ncv** $\times$ **ncv** $+ 6 \times$ **ncv** $+ 60$ and **lcomm** $\neq -1$.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

```
n = int32(100);
nev = int32(4);
ncv = int32(20);

h = 1/(double(n)+1);
rho = 10;
md = repmat(4*h, double(n), 1);
me = repmat(h, double(n-1), 1);

irevcm = int32(0);
resid = zeros(n,1);
v = zeros(n, ncv);
x = zeros(n, 1);
mx = zeros(n);

dd = 2/h;
dl = -1/h - rho/2;
du = -1/h + rho/2;
y = zeros(n,1);

[icomm, comm, ifail] = f12aa(n, nev, ncv);
[icomm, comm, ifail] = f12ad('REGULAR INVERSE', icomm, comm);
[icomm, comm, ifail] = f12ad('GENERALIZED', icomm, comm);

% Construct m and factorise
[md, me, info] = f07jd(md, me);

while (irevcm ~= 5)
  [irevcm, resid, v, x, mx, nshift, comm, icomm, ifail] = ...
    f12ab(irevcm, resid, v, x, mx, comm, icomm);
  if (irevcm == -1 || irevcm == 1)
    y(1) = dd*x(1) + du*x(2);
    for i = 2:n-1
      y(i) = dl*x(i-1) + dd*x(i) + du*x(i+1);
    end
    y(n) = dl*x(n-1) + dd*x(n);
    [x, info] = f07je(md, me, y);
  elseif (irevcm == 2)
    y(1) = 4*x(1) + x(2);
    for i=2:n-1
      y(i) = x(i-1) + 4*x(i) + x(i+1);
    end
    y(n) = x(n-1) + 4*x(n);
    x = h*y;
  elseif (irevcm == 4)
    [niter, nconv, ritzr, ritzi, rzest] = f12ae(icomm, comm);
    if (niter == 1)
      fprintf('\n');
    end
       fprintf('Iteration %2d No. converged = %d Norm of estimates =
%16.8e\n', niter, nconv, norm(rzest));
```

```
  end
end
[nconv, dr, di, z, v, comm, icomm, ifail] = f12ac(0, 0, resid, v, comm,
icomm)
```

```
Iteration  1 No. converged = 0 Norm of estimates =   5.56325675e+03
Iteration  2 No. converged = 0 Norm of estimates =   5.44836588e+03
Iteration  3 No. converged = 0 Norm of estimates =   5.30320774e+03
Iteration  4 No. converged = 0 Norm of estimates =   6.24234186e+03
Iteration  5 No. converged = 0 Norm of estimates =   7.15674705e+03
Iteration  6 No. converged = 0 Norm of estimates =   5.45460864e+03
Iteration  7 No. converged = 0 Norm of estimates =   6.43147571e+03
Iteration  8 No. converged = 0 Norm of estimates =   5.11241161e+03
Iteration  9 No. converged = 0 Norm of estimates =   7.19327824e+03
Iteration 10 No. converged = 1 Norm of estimates =   5.77945489e+03
Iteration 11 No. converged = 2 Norm of estimates =   4.73125738e+03
Iteration 12 No. converged = 3 Norm of estimates =   5.00078500e+03
nconv =
           4
dr =
   1.0e+04 *
    2.0383
    2.0339
    2.0265
    2.0163
di =
     0
     0
     0
     0
z =
     array elided
v =
     array elided
comm =
     array elided
icomm =
     array elided
ifail =
           0
```