NAG Toolbox

nag_sparseig_real_symm_init (f12fa)

1 Purpose

nag_sparseig_real_symm_init (f12fa) is a setup function in a suite of functions consisting of nag_sparseig_real_symm_init (f12fa), nag_sparseig_real_symm_iter (f12fb), nag_sparseig_real_symm_proc (f12fc), nag_sparseig_real_symm_option (f12fd) and nag_sparseig_real_symm_monit (f12fe). It is used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real symmetric matrices.

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

2 Syntax

```
[icomm, comm, ifail] = nag_sparseig_real_symm_init(n, nev, ncv)
[icomm, comm, ifail] = f12fa(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 symmetric. The suite can also be used to find selected eigenvalues/eigenvectors of smaller scale dense, real and symmetric problems.

nag_sparseig_real_symm_init (f12fa) is a setup function which must be called before nag_sparseig_real_symm_iter (f12fb), the reverse communication iterative solver, and before nag_sparseig_real_symm_option (f12fd), the options setting function. nag_sparseig_real_symm_proc (f12fc), is a post-processing function that must be called following a successful final exit from nag_sparseig_real_symm_iter (f12fb), while nag_sparseig_real_symm_monit (f12fe) 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 nag_sparseig_real_symm_option (f12fd), 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 11.1 in nag_sparseig_real_symm_option (f12fd).

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** – INTEGER

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** – INTEGER

The number of eigenvalues to be computed.

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

3: **ncv** – INTEGER

The number of Lanczos 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 increasing **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: $\mathbf{nev} < \mathbf{ncv} \leq \mathbf{n}$.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

1: **icomm**(**max**(**1**, $licomm$)) – INTEGER array

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

2: **comm**(**max**(**1**, $lcomm$)) – REAL (KIND=nag_wp) array

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

3: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

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, $\mathbf{ncv} \leq \mathbf{nev}$ or $\mathbf{ncv} > \mathbf{n}$.

ifail $= 4$

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

ifail $= 5$

On entry, $lcomm < 3 \times \mathbf{n} + \mathbf{ncv} \times \mathbf{ncv} + 8 \times \mathbf{ncv} + 60$ and $lcomm \neq -1$.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

This example solves $Ax = \lambda x$ in regular mode, where A is obtained from the standard central difference discretization of the Laplacian operator $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ on the unit square, with zero Dirichlet boundary conditions. Eigenvalues of smallest magnitude are selected.

9.1 Program Text

```
function f12fa_example

fprintf('f12fa example results\n\n');

nx  = nag_int(10);
n   = nx^2;
nev = nag_int(4);
ncv = nag_int(10);
imon = 0;

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

% Initialisation Step
[icomm, comm, ifail] = f12fa( ...
                              n, nev, ncv);

% Set Optional Parameters
[icomm, comm, ifail] = f12fd( ...
                              'SMALLEST MAGNITUDE', icomm, comm);

% Solve
while (irevcm ~= 5)
  [irevcm, resid, v, x, mx, nshift, comm, icomm, ifail] = ...
    f12fb( ...
           irevcm, resid, v, x, mx, comm, icomm);
```

```
  if (irevcm == 1 || irevcm == -1)
    x = f12fa_Ax(nx, x);
  elseif (irevcm == 4 && imon==1)
    [niter, nconv, ritz, rzest] = f12fe(icomm, comm);
    fprintf(['Iteration %2d, No. converged = %d, ', ...
             'norm of estimates = %10.2e\n'], ...
            niter, nconv, norm(rzest(1:nev),2));
  end
end

% Post-process to compute eigenvalues/vectors
sigma = 0;
[nconv, d, z, v, comm, icomm, ifail] = ...
  f12fc( ...
         sigma, resid, v, comm, icomm);

fprintf('Smallest %d Eigenvalues are:\n',nconv);
disp(d(1:nconv));

function [y] = f12fa_Ax(nx, x)

  y = zeros(nx*nx,1);

  h2 = 1/double((nx+1)^2);

  y(1:nx) = f12fa_Bx(nx, x(1:nx));
  y(1:nx) = y(1:nx) - x(nx+1:2*nx);

  for j=2:nx-1
    lo = (j-1)*nx +1;
    hi = j*nx;

    y(lo:hi) = f12fa_Bx(nx, x(lo:hi));
    y(lo:hi) = y(lo:hi) - x(lo-nx:lo-1) - x(hi+1:hi+nx);
  end

  lo = (nx-1)*nx +1;
  hi = nx*nx;
  y(lo:hi) = f12fa_Bx(nx, x(lo:hi));
  y(lo:hi) = y(lo:hi) - x(lo-nx:lo-1);
  y = y/h2;

function [y] = f12fa_Bx(nx,x)

  y = zeros(nx,1);

  dd = 4;
  dl = -1;
  du = -1;

  y(1) = dd*x(1) + du*x(2);
  for j=2:nx-1
    y(j) = dl*x(j-1) + dd*x(j) + du*x(j+1);
  end
  y(nx) = dl*x(nx-1) + dd*x(nx);
```

9.2 Program Results

```
     f12fa example results

Smallest 4 Eigenvalues are:
   19.6054
   48.2193
   48.2193
   76.8333
```