

NAG C Library Function Document

nag_real_symm_sparse_eigensystem_monit (f12fec)

1 Purpose

nag_real_symm_sparse_eigensystem_monit (f12fec) can be used to return additional monitoring information during computation. It is in a suite of functions consisting of nag_real_symm_sparse_eigensystem_monit (f12fec), nag_real_symm_sparse_eigensystem_init (f12fac), nag_real_symm_sparse_eigensystem_iter (f12fbc), nag_real_symm_sparse_eigensystem_sol (f12fcc) and nag_real_symm_sparse_eigensystem_option (f12fdc).

2 Specification

```
#include <nag.h>
#include <nagf12.h>
```

```
void nag_real_symm_sparse_eigensystem_monit (Integer *niter, Integer *nconv,
      double ritz[], double rzest[], Integer icomm[], double comm[])
```

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.

On an intermediate exit from nag_real_symm_sparse_eigensystem_iter (f12fbc) with **irevcm** = 4, nag_real_symm_sparse_eigensystem_monit (f12fec) may be called to return monitoring information on the progress of the Arnoldi iterative process. The information returned by nag_real_symm_sparse_eigensystem_monit (f12fec) is:

- the number of the current Arnoldi iteration;
- the number of converged eigenvalues at this point;
- the real and imaginary parts of the converged eigenvalues;
- the error bounds on the converged eigenvalues.

nag_real_symm_sparse_eigensystem_monit (f12fec) does not have an equivalent function from the ARPACK package which prints various levels of detail of monitoring information through an output channel controlled via a argument value (see Lehoucq *et al.* (1998) for details of ARPACK routines). nag_real_symm_sparse_eigensystem_monit (f12fec) should not be called at any time other than immediately following an **irevcm** = 4 return from nag_real_symm_sparse_eigensystem_iter (f12fbc).

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, Philadelphia

5 Arguments

- 1: **niter** – Integer * *Output*
On exit: the number of the current Arnoldi iteration.
- 2: **nconv** – Integer * *Output*
On exit: the number of converged eigenvalues so far.
- 3: **ritz**[*dim*] – double *Output*
Note: the dimension, *dim*, of the array **ritz** must be at least **nev** (see nag_real_symm_sparse_eigensystem_init (f12fac)).
On exit: the first **nconv** locations of the array **ritz** contain the real converged approximate eigenvalues.
- 4: **rzest**[*dim*] – double *Output*
Note: the dimension, *dim*, of the array **rzest** must be at least **nev** (see nag_real_symm_sparse_eigensystem_init (f12fac)).
On exit: the first **nconv** locations of the array **rzest** contain the Ritz estimates (error bounds) on the real **nconv** converged approximate eigenvalues.
- 5: **icomm**[*dim*] – Integer *Communication Array*
Note: the dimension, *dim*, of the array **icomm** must be at least $\max(1, \mathbf{lcomm})$ (see nag_real_symm_sparse_eigensystem_init (f12fac)).
icomm must remain unchanged.
- 6: **comm**[*dim*] – double *Communication Array*
Note: the dimension, *dim*, of the array **comm** must be at least $\max(1, \mathbf{lcomm})$ (see nag_real_symm_sparse_eigensystem_init (f12fac)).
comm must remain unchanged.

6 Error Indicators and Warnings

None.

7 Accuracy

A Ritz value, λ , is deemed to have converged if its Ritz estimate $\leq \mathbf{Tolerance} \times |\lambda|$. The default **Tolerance** used is the *machine precision* given by nag_machine_precision (X02AJC).

8 Further Comments

None.

9 Example

This example solves $Kx = \lambda K_G x$ using the **Buckling** option (see nag_real_symm_sparse_eigensystem_option (f12fdc), where K and K_G are obtained by the finite element method applied to the one-dimensional discrete Laplacian operator $\frac{\partial^2 u}{\partial x^2}$ on $[0, 1]$, with zero Dirichlet boundary conditions using piecewise linear elements. The shift, σ , is a real number, and the operator used in the Buckling iterative process is $OP = \text{inv}(K - \sigma K_G) \times K$ and $B = K$.

9.1 Program Text

```

/* nag_real_symm_sparse_eigensystem_monit (f12fec) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <nagf12.h>
#include <nagf16.h>
static void av(Integer, double *, double *);
static void my_dgtrf(Integer, double *, double *, double *,
                   double *, Integer *, Integer *);
static void my_dgtrs(Integer, double *, double *, double *,
                   double *, Integer *, double *, double *);

int main(void)
{
    /* Constants */
    Integer licomm=140, imon=1;

    /* Scalars */
    double estnrm, h, r1, r2, sigma;
    Integer exit_status, info, irevcm, j, lcomm, n, nconv, ncv;
    Integer nev, niter, nshift;
    /* Nag types */
    NagError fail;
    /* Arrays */
    double *dd=0, *dl=0, *du=0, *du2=0, *comm=0, *eigest=0;
    double *eigv=0, *resid=0, *v=0, *x2=0;
    Integer *icomm=0, *ipiv=0;
    /* Pointers */
    double *mx=0, *x=0, *y=0;

    exit_status = 0;
    INIT_FAIL(fail);

    Vprintf("nag_real_symm_sparse_eigensystem_monit (f12fec) Example Program "
           "Results\n");
    /* Skip heading in data file */
    Vscanf("%*[\n] ");

    /* Read values for nx, nev and cnv from data file. */
    Vscanf("%ld%ld%ld%*[\n] ", &n, &nev, &ncv);

    /* Allocate memory */
    lcomm = 3*n + ncv*ncv + 8*ncv + 60;
    if ( !(dd = NAG_ALLOC(n, double)) ||
         !(dl = NAG_ALLOC(n, double)) ||
         !(du = NAG_ALLOC(n, double)) ||
         !(du2 = NAG_ALLOC(n, double)) ||
         !(comm = NAG_ALLOC(lcomm, double)) ||
         !(eigv = NAG_ALLOC(ncv, double)) ||
         !(eigest = NAG_ALLOC(ncv, double)) ||
         !(resid = NAG_ALLOC(n, double)) ||
         !(v = NAG_ALLOC(n * ncv, double)) ||
         !(x2 = NAG_ALLOC(n, double)) ||
         !(icomm = NAG_ALLOC(licomm, Integer)) ||
         !(ipiv = NAG_ALLOC(n, Integer)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Initialise communication arrays for problem using

```

```

    nag_real_symm_sparse_eigensystem_init (f12fac). */
nag_real_symm_sparse_eigensystem_init(n, nev, ncv, icomm,
                                       licomm, comm, lcomm,
                                       &fail);

/* Select the problem type using
   nag_real_symm_sparse_eigensystem_option (f12fdc). */
nag_real_symm_sparse_eigensystem_option("generalized",
                                       icomm, comm, &fail);

/* Select the operating mode (Buckling) using
   nag_real_symm_sparse_eigensystem_option (f12fdc). */
nag_real_symm_sparse_eigensystem_option("buckling", icomm,
                                       comm, &fail);

/* Setup M and factorise */
h = 1.0 / (double) (n + 1);
r1 = 2.0 * h / 3.0;
r2 = h / 6.0;
sigma = 1.0;
for (j = 0; j <= n-1; ++j)
    {
        dd[j] = 2.0 / h - sigma * r1;
        dl[j] = -1.0 / h - sigma * r2;
        du[j] = dl[j];
    }
my_dgttrf(n, dl, dd, du, du2, ipiv, &info);

irevcm = 0;
REVCOMLOOP:
/* Repeated calls to reverse communication routine
   nag_real_symm_sparse_eigensystem_iter (f12fbc). */
nag_real_symm_sparse_eigensystem_iter(&irevcm, resid, v, &x, &y,
                                       &mx, &nshift, comm, icomm,
                                       &fail);

if (irevcm != 5)
    {
        if (irevcm == -1)
            {
                /* Perform  $y \leftarrow OP*x = inv[K-SIGMA*KG]*K*x.$  */
                av(n, x, x2);
                my_dgttrs(n, dl, dd, du, du2, ipiv, x2, y);
            }
        else if (irevcm == 1)
            {
                /* Perform  $y \leftarrow OP*x = inv[K-sigma*KG]*K*x.$  */
                my_dgttrs(n, dl, dd, du, du2, ipiv, mx, y);
            }
        else if (irevcm == 2)
            {
                /* Perform  $y \leftarrow K*x.$  */
                av(n, x, y);
            }
        else if (irevcm == 4 && imon == 1)
            {
                /* If imon=1, get monitoring information using
                   nag_real_symm_sparse_eigensystem_monit (f12fec). */
                nag_real_symm_sparse_eigensystem_monit(&niter, &nconv, eigv,
                                                       eigest, icomm, comm);

                /* Compute 2-norm of Ritz estimates using
                   nag_dge_norm (f16rac).*/
                nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, nev, 1, eigest,
                             nev, &estnrm, &fail);
                Vprintf("Iteration %3ld, ", niter);
                Vprintf(" No. converged = %3ld,", nconv);
                Vprintf(" norm of estimates = %16.8e\n", estnrm);
            }
        goto REVCOMLOOP;
    }
if (fail.code == NE_NOERROR)
    {

```

```

/* Post-Process using nag_real_symm_sparse_eigensystem_sol
   (f12fcc) to compute eigenvalues/vectors. */
nag_real_symm_sparse_eigensystem_sol(&nconv, eigv, v, sigma,
                                     resid, v, comm, icomm,
                                     &fail);

Vprintf("\n");
Vprintf(" The %4ld generalized Ritz values", nconv);
Vprintf(" closest to %8.4f are:\n\n", sigma);
for (j = 0; j <= nconv-1; ++j)
    {
        Vprintf("%8ld%5s%12.4f\n", j+1, "", eigv[j]);
    }
}
else
    {
        Vprintf(" Error from nag_real_symm_sparse_eigensystem_iter (f12fbc).",
               "\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
END:
if (dd) NAG_FREE(dd);
if (dl) NAG_FREE(dl);
if (du) NAG_FREE(du);
if (du2) NAG_FREE(du2);
if (comm) NAG_FREE(comm);
if (eigv) NAG_FREE(eigv);
if (eigest) NAG_FREE(eigest);
if (resid) NAG_FREE(resid);
if (v) NAG_FREE(v);
if (x2) NAG_FREE(x2);
if (icomm) NAG_FREE(icomm);
if (ipiv) NAG_FREE(ipiv);

return exit_status;
}

static void av(Integer n, double *v, double *y)
{
    /* Scalars */
    double h;
    Integer j;

    /* Function Body */
    h = (double) (n + 1);
    y[0] = h*(v[0] * 2.0 - v[1]);
    for (j = 1; j <= n - 2; ++j)
        {
            y[j] = h*(-v[j-1] + v[j] * 2.0 - v[j+1]);
        }
    y[n-1] = h*(-v[n-2] + v[n-1] * 2.0);
    return;
} /* av */

static void my_dgttrf(Integer n, double dl[], double d[],
                    double du[], double du2[], Integer ipiv[],
                    Integer *info)
{
    /* A simple C version of the Lapack routine dgttrf with argument
       checking removed */
    /* Scalars */
    double temp, fact;
    Integer i;
    /* Function Body */
    *info = 0;
    for (i = 0; i < n; ++i)
        {
            ipiv[i] = i;
        }
    for (i = 0; i < n - 2; ++i)
        {

```

```

    du2[i] = 0.0;
  }
  for (i = 0; i < n - 2; i++)
  {
    if (fabs(d[i]) >= fabs(dl[i]))
    {
      /* No row interchange required, eliminate dl[i]. */
      if (d[i] != 0.0)
      {
        fact = dl[i] / d[i];
        dl[i] = fact;
        d[i+1] = d[i+1] - fact * du[i];
      }
    }
    else
    {
      /* Interchange rows I and I+1, eliminate dl[I] */
      fact = d[i] / dl[i];
      d[i] = dl[i];
      dl[i] = fact;
      temp = du[i];
      du[i] = d[i+1];
      d[i+1] = temp - fact*d[i+1];
      du2[i] = du[i+1];
      du[i+1] = -fact * du[i+1];
      ipiv[i] = i + 1;
    }
  }
  if (n > 1)
  {
    i = n - 2;
    if (fabs(d[i]) >= fabs(dl[i]))
    {
      if (d[i] != 0.0)
      {
        fact = dl[i] / d[i];
        dl[i] = fact;
        d[i+1] = d[i+1] - fact * du[i];
      }
    }
    else
    {
      fact = d[i] / dl[i];
      d[i] = dl[i];
      dl[i] = fact;
      temp = du[i];
      du[i] = d[i+1];
      d[i+1] = temp - fact * d[i+1];
      ipiv[i] = i + 1;
    }
  }
  /* Check for a zero on the diagonal of U. */
  for (i = 0; i < n; ++i) {
    if (d[i] == 0.0)
    {
      *info = i;
      goto END;
    }
  }
  END:
  return;
}

static void my_dgttrs(Integer n, double dl[], double d[],
                    double du[], double du2[], Integer ipiv[],
                    double b[], double y[])
{
  /* A simple C version of the Lapack routine dgttrs with argument
   checking removed, the number of right-hand-sides=1, Trans='N' */
  /* Scalars */
  Integer i, ip;

```

```

double temp;
/* Solve L*x = b. */
for (i = 0; i <= n - 1; ++i)
  {
    y[i] = b[i];
  }
for (i = 0; i < n - 1; ++i)
  {
    ip = ipiv[i];
    temp = y[i+1-ip+i] - dl[i]*y[ip];
    y[i] = y[ip];
    y[i+1] = temp;
  }
/* Solve U*x = b. */
y[n-1] = y[n-1] / d[n-1];
if (n > 1) {
  y[n-2] = (y[n-2] - du[n-2]*y[n-1])/d[n-2];
}
for (i = n - 3; i >= 0; --i)
  {
    y[i] = (y[i]-du[i]*y[i+1]-du2[i]*y[i+2])/d[i];
  }
return;
}

```

9.2 Program Data

nag_real_symm_sparse_eigensystem_monit (f12fec) Example Program Data
 100 4 10 : Values for n, nev and ncv

9.3 Program Results

nag_real_symm_sparse_eigensystem_monit (f12fec) Example Program Results
 Iteration 1, No. converged = 0, norm of estimates = 2.05343313e-06
 Iteration 2, No. converged = 2, norm of estimates = 6.07599403e-11
 Iteration 3, No. converged = 3, norm of estimates = 5.26525802e-15

The 4 generalized Ritz values closest to 1.0000 are:

1	9.8704
2	39.4912
3	88.8909
4	158.1175
