NAG Library Function Document

nag_real_sparse_eigensystem_sol (f12acc)

Note: *this function uses* **optional arguments** *to define choices in the problem specification. If you wish to use* default *settings for all of the optional arguments, then the option setting function nag_real_sparse_eigensystem_option (f12adc) need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 in nag_real_sparse_eigensystem_option (f12adc) for a detailed description of the specification of the optional arguments*.

1 Purpose

nag_real_sparse_eigensystem_sol (f12acc) is a post-processing function that must be called following a final exit from nag_real_sparse_eigensystem_iter (f12abc). These are part of a suite of functions for the solution of real sparse eigensystems. The suite also includes nag_real_sparse_eigensystem_init (f12aac), nag_real_sparse_eigensystem_option (f12adc) and nag_real_sparse_eigensystem_monit (f12aec).

2 Specification

```
#include <nag.h>
#include <nagf12.h>
void nag_real_sparse_eigensystem_sol (Integer *nconv, double dr[],
     double di[], double z[], double sigmar, double sigmai,
     const double resid[], double v[], double comm[], Integer icomm[],
     NagError *fail)
```

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.

Following a call to nag_real_sparse_eigensystem_iter (f12abc), nag_real_sparse_eigensystem_sol (f12acc) returns the converged approximations to eigenvalues and (optionally) the corresponding approximate eigenvectors and/or an orthonormal basis for the associated approximate invariant subspace. The eigenvalues (and eigenvectors) are selected from those of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices. There is negligible additional cost to obtain eigenvectors; an orthonormal basis is always computed, but there is an additional storage cost if both are requested.

nag_real_sparse_eigensystem_sol (f12acc) is based on the function **dneupd** from the ARPACK package, which uses the Implicitly Restarted Arnoldi iteration method. The method is described in Lehoucq and Sorensen (1996) and Lehoucq (2001) while its use within the ARPACK software is described in great detail in Lehoucq *et al.* (1998). An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices is provided in Lehoucq and Scott (1996). This suite of functions offers the same functionality as the ARPACK software for real nonsymmetric problems, but the interface design is quite different in order to make the option setting clearer and to simplify some of the interfaces.

nag_real_sparse_eigensystem_sol (f12acc), is a post-processing function that must be called following a successful final exit from nag_real_sparse_eigensystem_iter (f12abc). nag_real_sparse_eigensystem_sol (f12acc) uses data returned from nag_real_sparse_eigensystem_iter (f12abc) and options, set either by default or explicitly by calling nag_real_sparse_eigensystem_option (f12adc), to return the converged approximations to selected eigenvalues and (optionally):

– the corresponding approximate eigenvectors;

– an orthonormal basis for the associated approximate invariant subspace;

– both.

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 Arguments

1: **nconv** – Integer * *Output*

On exit: the number of converged eigenvalues as found by nag_real_sparse_eigensystem_iter (f12abc).

2: **dr**[dim] – double *Output*

Note: the dimension, dim, of the array **dr** must be at least **nev** (see nag_real_sparse_eigensystem_init (f12aac)).

On exit: the first **nconv** locations of the array **dr** contain the real parts of the converged approximate eigenvalues.

3: **di**[dim] – double *Output*

Note: the dimension, dim, of the array **di** must be at least **nev** (see nag_real_sparse_eigensystem_init (f12aac)).

On exit: the first **nconv** locations of the array **di** contain the imaginary parts of the converged approximate eigenvalues.

4: **z**[**n** × (**nev** + **1**)] – double *Output*

On exit: if the default option **Vectors** = RITZ (see nag_real_sparse_eigensystem_option (f12adc)) has been selected then **z** contains the final set of eigenvectors corresponding to the eigenvalues held in **dr** and **di**. The complex eigenvector associated with the eigenvalue with positive imaginary part is stored in two consecutive array segments. The first segment holds the real part of the eigenvector and the second holds the imaginary part. The eigenvector associated with the eigenvalue with negative imaginary part is simply the complex conjugate of the eigenvector associated with the positive imaginary part.

For example, the first eigenvector has real parts stored in locations **z**[$i-1$], for $i = 1, 2, \ldots,$ **n** and imaginary parts stored in **z**[$i-1$], for $i =$ **n** $+ 1,$ 2**n**.

5: **sigmar** – double *Input*

On entry: if one of the **Shifted Inverse Real** modes have been selected then **sigmar** contains the real part of the shift used; otherwise **sigmar** is not referenced.

6: **sigmai** – double *Input*

On entry: if one of the **Shifted Inverse Real** modes have been selected then **sigmai** contains the imaginary part of the shift used; otherwise **sigmai** is not referenced.

7: **resid**[dim] – const double *Input*

Note: the dimension, dim, of the array **resid** must be at least **n** (see nag_real_sparse_eigensystem_init (f12aac)).

On entry: must not be modified following a call to nag_real_sparse_eigensystem_iter (f12abc) since it contains data required by nag_real_sparse_eigensystem_sol (f12acc).

8: **v**[**n** × **ncv**] – double *Input/Output*

The ith element of the jth basis vector is stored in location **v**[**n** × $(j-1)+i-1$], for $i = 1, 2, \ldots,$ **n** and $j = 1, 2, \ldots,$ **ncv**.

On entry: the **ncv** sections of **v**, of length n, contain the Arnoldi basis vectors for OP as constructed by nag_real_sparse_eigensystem_iter (f12abc).

On exit: if the option **Vectors** = SCHUR has been set, or the option **Vectors** = RITZ has been set and a separate array **z** has been passed (i.e., **z** does not equal **v**), then the first **nconv** sections of **v**, of length n, will contain approximate Schur vectors that span the desired invariant subspace.

9: **comm**[dim] – double *Communication Array*

Note: the dimension, dim, of the array **comm** must be at least max(1, **lcomm**) (see nag_real_sparse_eigensystem_init (f12aac)).

On initial entry: must remain unchanged from the prior call to nag_real_sparse_eigensystem_iter (f12abc).

On exit: contains data on the current state of the solution.

10: **icomm**[dim] – Integer *Communication Array*

Note: the dimension, dim, of the array **icomm** must be at least max(1, **licomm**) (see nag_real_sparse_eigensystem_init (f12aac)).

On initial entry: must remain unchanged from the prior call to nag_real_sparse_eigensystem_iter (f12abc).

On exit: contains data on the current state of the solution.

11: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INITIALIZATION

Either the solver function has not been called prior to the call of this function or a communication array has become corrupted.

NE_INTERNAL_EIGVEC_FAIL

In calculating eigenvectors, an internal call returned with an error. Please contact NAG.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_INVALID_OPTION

On entry, **Vectors** = SELECT, but this is not yet implemented.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_RITZ_COUNT

Got a different count of the number of converged Ritz values than the value passed to it through the argument **icomm**: number counted = ⟨*value*⟩, number expected = ⟨*value*⟩. This usually indicates that a communication array has been altered or has become corrupted between calls to nag_real_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc).

NE_SCHUR_EIG_FAIL

During calculation of a real Schur form, there was a failure to compute ⟨*value*⟩ eigenvalues in a total of ⟨*value*⟩ iterations.

NE_SCHUR_REORDER

The computed Schur form could not be reordered by an internal call. This function returned with **fail.code** = ⟨*value*⟩. Please contact NAG.

NE_ZERO_EIGS_FOUND

The number of eigenvalues found to sufficient accuracy, as communicated through the argument **icomm**, is zero. You should experiment with different values of **nev** and **ncv**, or select a different computational mode or increase the maximum number of iterations prior to calling nag_real_sparse_eigensystem_iter (f12abc).

7 Accuracy

The relative accuracy of a Ritz value, λ, is considered acceptable if its Ritz estimate $\leq$ **Tolerance** $\times |\lambda|$. The default **Tolerance** used is the ***machine precision*** given by nag_machine_precision (X02AJC).

8 Parallelism and Performance

nag_real_sparse_eigensystem_sol (f12acc) is not threaded by NAG in any implementation.

nag_real_sparse_eigensystem_sol (f12acc) 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 function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example solves $Ax = \lambda Bx$ in regular-invert mode, where A and B are obtained from the standard central difference discretization of the one-dimensional convection-diffusion operator $\frac{d^2u}{dx^2} + \rho\frac{du}{dx}$ on $[0, 1]$, with zero Dirichlet boundary conditions.

10.1 Program Text

```
/* nag_real_sparse_eigensystem_sol (f12acc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <nagf12.h>
#include <nagf16.h>

static void av(Integer, double, double *, double *);
static void mv(Integer, double *, double *);
static void my_dpttrf(Integer, double *, double *, Integer *);
static void my_dpttrs(Integer, double *, double *, double *);

int main(void)
{
  /* Constants */
  Integer  licomm = 140, imon = 0;
  /* Scalars */
  double   estnrm, h, rho, sigmai = 0.0, sigmar = 0.0;
  Integer  exit_status, info, irevcm, j, lcomm, n, nconv, ncv;
  Integer  nev, niter, nshift, nx;
  /* Nag types */
  NagError fail;
  /* Arrays */
  double   *comm = 0, *eigvr = 0, *eigvi = 0, *eigest = 0, *md = 0, *me = 0;
  double   *resid = 0, *v = 0;
  Integer  *icomm = 0;
  /* Pointers */
  double   *mx = 0, *x = 0, *y = 0;

  exit_status = 0;
  INIT_FAIL(fail);

  printf(
          "nag_real_sparse_eigensystem_sol (f12acc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read problem parameter values from data file. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%lf%*[^\n] ", &nx, &nev, &ncv, &rho);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%lf%*[^\n] ", &nx, &nev, &ncv, &rho);
#endif
  n = nx * nx;
  lcomm = 3*n + 3*ncv*ncv + 6*ncv + 60;
  /* Allocate memory */
  if (!(comm = NAG_ALLOC(lcomm, double)) ||
      !(eigvr = NAG_ALLOC(ncv, double)) ||
      !(eigvi = NAG_ALLOC(ncv, double)) ||
      !(eigest = NAG_ALLOC(ncv, double)) ||
      !(md = NAG_ALLOC(n, double)) ||
      !(me = NAG_ALLOC(n, double)) ||
```

```
      !(resid = NAG_ALLOC(n, double)) ||
      !(v = NAG_ALLOC(n * ncv, double)) ||
      !(icomm = NAG_ALLOC(licomm, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Initialise communication arrays for problem using
     nag_real_sparse_eigensystem_init (f12aac). */
  nag_real_sparse_eigensystem_init(n, nev, ncv, icomm, licomm, comm,
                                   lcomm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_real_sparse_eigensystem_init (f12aac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /* Set the mode. */
  /* Select the mode using
     nag_real_sparse_eigensystem_option (f12adc). */
  nag_real_sparse_eigensystem_option("REGULAR INVERSE", icomm, comm,
                                     &fail);
  /* Select the problem type using
     nag_real_sparse_eigensystem_option (f12adc). */
  nag_real_sparse_eigensystem_option("GENERALIZED", icomm, comm, &fail);

  /* Construct M, and factorize using my_dpttrf. */
  h = 1.0 / (double)(n + 1);
  for (j = 0; j <= n - 2; ++j)
    {
      md[j] = h * 4.0;
      me[j] = h;
    }
  md[n - 1] = h * 4.0;

  my_dpttrf(n, md, me, &info);

  irevcm = 0;
 REVCOMLOOP:
  /* repeated calls to reverse communication routine
     nag_real_sparse_eigensystem_iter (f12abc). */
  nag_real_sparse_eigensystem_iter(&irevcm, resid, v, &x, &y, &mx,
                                   &nshift, comm, icomm, &fail);
  if (irevcm != 5)
    {
      if (irevcm == -1 || irevcm == 1)
        {
          /* Perform  y <--- OP*x = inv[M]*A*x using my_dpttrs. */
          av(nx, rho, x, y);
          my_dpttrs(n, md, me, y);
        }
      else if (irevcm == 2)
        {
          /* Perform  y <--- M*x. */
          mv(nx, x, y);
        }
      else if (irevcm == 4 && imon == 1)
        {
          /* If imon=1, get monitoring information using
             nag_real_sparse_eigensystem_monit (f12aec). */
          nag_real_sparse_eigensystem_monit(&niter, &nconv, eigvr,
                                            eigvi, 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);
          printf("Iteration %3"NAG_IFMT", ", niter);
          printf(" No. converged = %3"NAG_IFMT",", nconv);
```

```
          printf(" norm of estimates = %17.8e\n", estnrm);
        }
      goto REVCOMLOOP;
    }
  if (fail.code == NE_NOERROR)
    {
      /* Post-Process using nag_real_sparse_eigensystem_sol (f12acc)
         to compute eigenvalues/vectors. */
      nag_real_sparse_eigensystem_sol(&nconv, eigvr, eigvi, v, sigmar,
                                      sigmai, resid, v, comm, icomm,
                                      &fail);
      /* Print computed eigenvalues. */
      printf("\n  The %4"NAG_IFMT" generalized", nconv);
      printf(" Ritz values of largest magnitude are:\n\n");
      for (j = 0; j <= nconv-1; ++j)
        {
          printf("%8"NAG_IFMT"%5s( %12.4f ,%12.4f )\n", j+1, "",
                  eigvr[j], eigvi[j]);
        }
    }
  else
    {
      printf(
              " Error from nag_real_sparse_eigensystem_iter (f12abc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(comm);
  NAG_FREE(eigvr);
  NAG_FREE(eigvi);
  NAG_FREE(eigest);
  NAG_FREE(md);
  NAG_FREE(me);
  NAG_FREE(resid);
  NAG_FREE(v);
  NAG_FREE(icomm);
  return exit_status;
}

static void av(Integer nx, double rho, double *v, double *y)
{
  /* Scalars */
  double  dd, dl, du, h, s;
  Integer j, n;
  /* Function Body */
  n = nx * nx;
  h = 1.0 / (double)(n + 1);
  s = rho / 2.0;
  dd = 2.0 / h;
  dl = -1.0 / h - s;
  du = -1.0 / h + s;
  y[0] = dd * v[0] + du * v[1];
  for (j = 1; j <= n - 2; ++j)
    {
      y[j] = dl * v[j-1] + dd * v[j] + du * v[j+1];
    }
  y[n-1] = dl * v[n-2] + dd * v[n-1];
  return;
} /* av */

static void mv(Integer nx, double *v, double *y)
{
  /* Scalars */
  double  h;
  Integer j, n;
  /* Function Body */
  n = nx * nx;
  h = 1. / (double)(n + 1);
  y[0] = h*(v[0] * 4. + v[1]);
```

```
  for (j = 1; j <= n - 2; ++j)
    {
      y[j] = h*(v[j-1] + v[j] * 4. + v[j+1]);
    }
  y[n-1] = h*(v[n-2] + v[n-1] * 4.);
  return;
} /* mv */

static void my_dpttrf(Integer n, double d[], double e[], Integer *info)
{
  /* A simple C version of the Lapack routine dpttrf with argument
     checking removed */
  /* Scalars */
  double  ei;
  Integer i;
  /* Function Body */
  *info = 0;
  for (i = 0; i < n-1; ++i)
    {
      if (d[i] <= 0.0)
        {
          *info = i+1;
          goto END_DPTTRF;
        }
      ei = e[i];
      e[i] = ei/d[i];
      d[i+1] = d[i+1] - e[i]*ei;
    }
  if (d[n-1] <= 0.0)
    {
      *info = n;
    }
 END_DPTTRF:
  return;
}

static void my_dpttrs(Integer n, double d[], double e[], double b[])
{
  /* A simple C version of the Lapack routine dpttrs with argument
     checking removed and nrhs=1 */
  /* Scalars */
  Integer i;
  /* Function Body */
  for (i = 1; i < n; ++i)
    {
      b[i] = b[i] - b[i-1]*e[i-1];
    }
  b[n-1] = b[n-1]/d[n-1];
  for (i = n-2; i >= 0; --i)
    {
      b[i] = b[i]/d[i] - b[i+1]*e[i];
    }
  return;
}
```

10.2 Program Data

```
nag_real_sparse_eigensystem_sol (f12acc) Example Program Data
 10  4  20  10.0 : Values for nx, nev, ncv, rho
```

10.3 Program Results

```
nag_real_sparse_eigensystem_sol (f12acc) Example Program Results

  The    4 generalized Ritz values of largest magnitude are:

       1     (   20383.0384 ,      0.0000 )
       2     (   20338.7563 ,      0.0000 )
       3     (   20265.2844 ,      0.0000 )
       4     (   20163.1142 ,      0.0000 )
```