NAG Library Function Document

nag_eigen_real_symm_sparse_arnoldi (f02fkc)

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, you need only read Sections 1 to 10 of this document. If, however, you wish to reset some or all of the settings this must be done by calling the option setting function nag_real_symm_sparse_eigensystem_option (f12fdc) from the user-supplied function* ***option****. Please refer to Section 11 for a detailed description of the specification of the optional arguments.*

1 Purpose

nag_eigen_real_symm_sparse_arnoldi (f02fkc) computes selected eigenvalues and eigenvectors of a real sparse symmetric matrix.

2 Specification

```
#include <nag.h>
#include <nagf02.h>
void nag_eigen_real_symm_sparse_arnoldi (Integer n, Integer nnz,
     const double a[], const Integer irow[], const Integer icol[],
     Integer nev, Integer ncv, double sigma,
     void (*monit)(Integer ncv, Integer niter, Integer nconv,
          const double w[], const double rzest[], Integer *istat,
          Nag_Comm *comm),
     void (*option)(Integer icom[], double com[], Integer *istat,
          Nag_Comm *comm),
     Integer *nconv, double w[], double v[], Integer pdv, double resid[],
     Nag_Comm *comm, NagError *fail)
```

3 Description

nag_eigen_real_symm_sparse_arnoldi (f02fkc) computes selected eigenvalues and the corresponding right eigenvectors of a real sparse symmetric matrix A:

$$Av_i = \lambda_i v_i.$$

A specified number, n_{ev}, of eigenvalues λ_i, or the shifted inverses $\mu_i = 1/(\lambda_i - \sigma)$, may be selected either by largest or smallest modulus, largest or smallest value, or, largest and smallest values (both ends). Convergence is generally faster when selecting larger eigenvalues, smaller eigenvalues can always be selected by choosing a zero inverse shift ($\sigma = 0.0$). When eigenvalues closest to a given value are required then the shifted inverses of largest magnitude should be selected with shift equal to the required value.

The sparse matrix A is stored in symmetric coordinate storage (SCS) format. See Section 2.1.2 in the f11 Chapter Introduction.

nag_eigen_real_symm_sparse_arnoldi (f02fkc) uses an implicitly restarted Arnoldi (Lanczos) iterative method to converge approximations to a set of required eigenvalues and corresponding eigenvectors. Further algorithmic information is given in Section 9 while a fuller discussion is provided in the f12 Chapter Introduction. If shifts are to be performed then operations using shifted inverse matrices are performed using a direct sparse solver.

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

2: **nnz** – Integer *Input*

On entry: the dimension of the array **a**.The number of nonzero elements in the lower triangular part of the matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n}+1)/2$.

3: **a**[**nnz**] – const double *Input*

On entry: the array of nonzero elements of the lower triangular part of the n by n symmetric matrix A.

4: **irow**[**nnz**] – const Integer *Input*

5: **icol**[**nnz**] – const Integer *Input*

On entry: the row and column indices of the elements supplied in array **a**.

If $\mathbf{irow}[k-1] = i$ and $\mathbf{icol}[k-1] = j$ then A_{ij} is stored in $\mathbf{a}[k-1]$. **irow** does not need to be ordered, an internal call to nag_sparse_sym_sort (f11zbc) forces the correct ordering.

Constraint:

irow and **icol** must satisfy these constraints:$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ and $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$, for $i = 0, 1, \ldots, \mathbf{nnz}-1$.

6: **nev** – Integer *Input*

On entry: the number of eigenvalues to be computed.

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

7: **ncv** – Integer *Input*

On entry: the dimension of the array **w**. 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 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 computation time is problem dependent and must be determined empirically.

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

8: **sigma** – double *Input*

On entry: if the **Shifted Inverse** mode has been selected then **sigma** contains the real shift used; otherwise **sigma** is not referenced. This mode can be selected by setting the appropriate options in the user-supplied function **option**.

9: **monit** – function, supplied by the user *External Function*

monit is used to monitor the progress of nag_eigen_real_symm_sparse_arnoldi (f02fkc). **monit** may be specified as **NULLFN** if no monitoring is actually required. **monit** is called after the solution of each eigenvalue sub-problem and also just prior to return from nag_eigen_real_symm_sparse_arnoldi (f02fkc).

The specification of **monit** is:

```
void monit (Integer ncv, Integer niter, Integer nconv,
     const double w[], const double rzest[], Integer *istat,
     Nag_Comm *comm)
```

1: **ncv** – Integer *Input*

On entry: the dimension of the arrays **w** and **rzest**. The number of Arnoldi basis vectors used during the computation.

2: **niter** – Integer *Input*

On entry: the number of the current Arnoldi iteration.

3: **nconv** – Integer *Input*

On entry: the number of converged eigenvalues so far.

4: **w**[**ncv**] – const double *Input*

On entry: the first **nconv** elements of **w** contain the converged approximate eigenvalues.

5: **rzest**[**ncv**] – const double *Input*

On entry: the first **nconv** elements of **rzest** contain the Ritz estimates (error bounds) on the converged approximate eigenvalues.

6: **istat** – Integer * *Input/Output*

On entry: set to zero.

On exit: if set to a nonzero value nag_eigen_real_symm_sparse_arnoldi (f02fkc) returns immediately with **fail**.**code** = NE_USER_STOP.

7: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **monit**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_eigen_real_symm_sparse_arnoldi (f02fkc) you may allocate memory and initialize these pointers with various quantities for use by **monit** when called from nag_eigen_real_symm_sparse_arnoldi (f02fkc) (see Section 3.2.1.1 in the Essential Introduction).

10: **option** – function, supplied by the user *External Function*

You can supply non-default options to the Arnoldi eigensolver by repeated calls to nag_real_symm_sparse_eigensystem_option (f12fdc) from within **option**. (Please note that it is only necessary to call nag_real_symm_sparse_eigensystem_option (f12fdc); no call to nag_real_symm_sparse_eigensystem_init (f12fac) is required from within **option**.) For example, you can set the mode to **Shifted Inverse**, you can increase the **Iteration Limit** beyond its default and you can print varying levels of detail on the iterative process using **Print Level**.

If only the default options (including that the eigenvalues of largest magnitude are sought) are to be used then **option** may be specified as **NULLFN**. See Section 10 for an example of using **option** to set some non-default options.

The specification of **option** is:

```
void option (Integer icom[], double com[], Integer *istat,
     Nag_Comm *comm)
```

1: **icom**[**140**] – Integer *Communication Array*

On entry: contains details of the default option set. This array must be passed as argument **icomm** in any call to nag_real_symm_sparse_eigensystem_option (f12fdc).

On exit: contains data on the current options set which may be altered from the default set via calls to nag_real_symm_sparse_eigensystem_option (f12fdc).

2: **com**[**60**] – double *Communication Array*

On entry: contains details of the default option set. This array must be passed as argument **comm** in any call to nag_real_symm_sparse_eigensystem_option (f12fdc).

On exit: contains data on the current options set which may be altered from the default set via calls to nag_real_symm_sparse_eigensystem_option (f12fdc).

3: **istat** – Integer * *Input/Output*

On entry: set to zero.

On exit: if set to a nonzero value nag_eigen_real_symm_sparse_arnoldi (f02fkc) returns immediately with **fail**.**code** = NE_USER_STOP.

4: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **option**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_eigen_real_symm_sparse_arnoldi (f02fkc) you may allocate memory and initialize these pointers with various quantities for use by **option** when called from nag_eigen_real_symm_sparse_arnoldi (f02fkc) (see Section 3.2.1.1 in the Essential Introduction).

11: **nconv** – Integer * *Output*

On exit: the number of converged approximations to the selected eigenvalues. On successful exit, this will normally be **nev**.

12: **w**[**ncv**] – double *Output*

On exit: the first **nconv** elements contain the converged approximations to the selected eigenvalues.

13: **v**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **v** must be at least **pdv** × **ncv**.

On exit: contains the eigenvectors associated with the eigenvalue λ_i, for $i = 1, 2, \ldots,$ **nconv** (stored in **w**). For eigenvalue, λ_j, the corresponding eigenvector is stored in **v**$[(j-1) \times$ **pdv** $+ i - 1]$, for $i = 1, 2, \ldots, n$.

14: **pdv** – Integer *Input*

On entry: the stride separating, in the array **v**, the elements of a real eigenvector from the corresponding elements of the next eigenvector.

Constraint: **pdv** $\geq$ **n**.

15: **resid**[**nev**] – double *Output*

On exit: the residual $\|Aw_i - \lambda_i w_i\|_2$ for the estimates to the eigenpair λ_i and w_i is returned in **resid**$[i-1]$, for $i = 1, 2, \ldots,$ **nconv**.

16: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

17: **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 $\langle value \rangle$ had an illegal value.

NE_BOTH_ENDS_1

The option **Both Ends** has been set but only 1 eigenvalue is requested.

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** > 0.

On entry, **nev** $= \langle value \rangle$.
Constraint: **nev** > 0.

On entry, **nnz** $= \langle value \rangle$.
Constraint: **nnz** > 0.

NE_INT_2

On entry, **ncv** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **ncv** $\leq$ **n**.

On entry, **ncv** $= \langle value \rangle$ and **nev** $= \langle value \rangle$.
Constraint: **ncv** $>$ **nev**.

On entry, **nev** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **nev** $< (\mathbf{n} - 1)$.

On entry, **nnz** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **nnz** $\leq \mathbf{n} \times (\mathbf{n}+1)/2$.

On entry, **pdv** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdv** $\geq$ **n**.

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.

A serious error, code ⟨*value*⟩, has occurred in an internal call to ⟨*value*⟩. Check all function calls and array sizes. If the call is correct then please contact NAG for assistance.

NE_INVALID_OPTION

The maximum number of iterations, through the optional argument **Iteration Limit**, has been set to a non-positive value.

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_SINGULAR

On entry, the matrix $(A - \sigma I)$ is numerically singular and could not be inverted. Try perturbing the value of σ.

NE_SPARSE_COL

On entry, for $i = \langle value \rangle$, **icol**$[i-1] = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$.
Constraint: $1 \leq$ **icol**$[i-1] \leq$ **irow**$[i-1]$.

NE_SPARSE_ROW

On entry, for $i = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$.
Constraint: $1 \leq$ **irow**$[i-1] \leq$ **n**.

NE_TOO_MANY_ITER

The maximum number of iterations has been reached.
The maximum number of iterations $= \langle value \rangle$.
The number of converged eigenvalues $= \langle value \rangle$.
See the function document for further details.

NE_USER_STOP

User requested termination in **monit**, **istat** $= \langle value \rangle$.

User requested termination in **option**, **istat** $= \langle value \rangle$.

7 Accuracy

The relative accuracy of a Ritz value (eigenvalue approximation), λ , is considered acceptable if its Ritz estimate $\leq$ **Tolerance** $\times \lambda$. The default value for **Tolerance** is the ***machine precision*** given by nag_machine_precision (X02AJC). The Ritz estimates are available via the **monit** function at each iteration in the Arnoldi process, or can be printed by setting option **Print Level** to a positive value.

8 Parallelism and Performance

nag_eigen_real_symm_sparse_arnoldi (f02fkc) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_eigen_real_symm_sparse_arnoldi (f02fkc) 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

nag_eigen_real_symm_sparse_arnoldi (f02fkc) calls functions based on the ARPACK suite in Chapter f12. These functions use an implicitly restarted Lanczos iterative method to converge to approximations to a set of required eigenvalues (see the f12 Chapter Introduction).

In the default **Regular** mode, only matrix-vector multiplications are performed using the sparse matrix A during the Lanczos process; nag_sparse_sym_matvec (f11xec) can be used to perform this task. Each iteration is therefore cheap computationally, relative to the alternative, **Shifted Inverse**, mode described below. It is most efficient (i.e., the total number of iterations required is small) when the eigenvalues of largest magnitude are sought and these are distinct.

Although there is an option for returning the smallest eigenvalues using this mode (see **Smallest Magnitude** option), the number of iterations required for convergence will be far greater or the method may not converge at all. However, where convergence is achieved, **Regular** mode may still prove to be the most efficient since no inversions are required. Where smallest eigenvalues are sought and **Regular** mode is not suitable, or eigenvalues close to a given real value are sought, the **Shifted Inverse** mode should be used.

If the **Shifted Inverse** mode is used (via a call to nag_real_symm_sparse_eigensystem_option (f12fdc) in **option**) then the matrix $A - \sigma I$ is used in linear system solves by the Lanczos process. This is first factorized internally using a direct sparse LDL^{T} factorization under the assumption that the matrix is indefinite. If the factorization determines that the matrix is numerically singular then the function exits with an error. In this situation it is normally sufficient to perturb σ by a small amount and call nag_eigen_real_symm_sparse_arnoldi (f02fkc) again. After successful factorization, subsequent solves are performed by backsubstitution using the sparse factorization.

Finally, nag_eigen_real_symm_sparse_arnoldi (f02fkc) transforms the eigenvectors. Each eigenvector w is normalized so that $\|w\|_2 = 1$.

The monitoring function **monit** provides some basic information on the convergence of the Lanczos iterations. Much greater levels of detail on the Lanczos process are available via option **Print Level**. If this is set to a positive value then information will be printed, by default, to standard output. The destination of monitoring information can be changed using the **Monitoring** option.

10 Example

This example solves $Ax = \lambda x$ in **Shifted Inverse** mode, where A is obtained from the standard central difference discretization of the one-dimensional Laplacian operator $\frac{\partial^2 u}{\partial x^2}$ on $[0, 1]$, with zero Dirichlet boundary conditions.

10.1 Program Text

```
/* nag_eigen_real_symm_sparse_arnoldi (f02fkc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf02.h>
#include <nagf12.h>
#include <nagx02.h>
#include <nagx04.h>
```

```
/* User-defined Functions */
#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL myoption(Integer icomm[], double com[], Integer *istat,
                              Nag_Comm *comm);

static void NAG_CALL mymonit(Integer ncv, Integer niter, Integer nconv,
                             const double w[], const double rzest[],
                             Integer *istat, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  double   h2, sigma;
  Integer  exit_status = 0;
  Integer  fileid, fmode, i, imon, j, k, lo, maxit, mode;
  Integer  n, nconv, ncv, nev, nnz, nx, prtlvl, tdv;

  /* Local Arrays */
  double   *w = 0, *a = 0, *resid = 0, *v = 0;
  double   user[1];
  Integer  *icol = 0, *irow = 0;
  Integer  iuser[5];
  const char *filename = "f02fkce.monit";

  /* Nag Types */
  Nag_Comm comm;
  NagError fail;

  INIT_FAIL(fail);

  comm.user = user;
  comm.iuser = iuser;
  user[0] = 0.0;
  iuser[0] = 0;

  /* Output preamble */
  printf(" nag_eigen_real_symm_sparse_arnoldi (f02fkc) ");
  printf("Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in problem size and parameters */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]%"NAG_IFMT"", &nx, &nev, &ncv);
#else
  scanf("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]%"NAG_IFMT"", &nx, &nev, &ncv);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]%lf%*[^\n]", &sigma);
#else
  scanf("%*[^\n]%lf%*[^\n]", &sigma);
#endif

  n = nx * nx;
  nnz = 3 * n - 2*nx;
  tdv = n;

  if (!(resid  = NAG_ALLOC((ncv),   double)) ||
      !(a      = NAG_ALLOC((nnz),   double)) ||
```

```
      !(icol   = NAG_ALLOC((nnz), Integer)) ||
      !(irow   = NAG_ALLOC((nnz),   Integer)) ||
      !(w      = NAG_ALLOC((ncv),   double)) ||
      !(v      = NAG_ALLOC((tdv)*(ncv), double))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Construct A in sparse (SCS) format where:
   *   A_{i,i}   = 4/(h*h)
   *   A_{i+1,i} = -1/(h*h)
   *   A_{i+nx,i} = -1/(h*h)
   */
  h2 = 1.0/(double)((nx+1)*(nx+1));

  /* Main Diagonal of A */
  k = 0;
  for (i = 1; i <= n; i++) {
    irow[k] = i;
    icol[k] = i;
    a[k] = 4.0/h2;
    k++;
  }

  /* First subdiagonal of A. */
  for (i = 1; i <= nx; i++) {
    lo = (i-1)*nx;
    for (j = lo + 1; j <= lo + nx - 1; j++) {
      irow[k] = j + 1;
      icol[k] = j;
      a[k] = -1.0/h2;
      k++;
    }
  }

  /* nx-th subdiagonal of A. */
  for (i = 1; i < nx; i++) {
    lo = (i-1)*nx;
    for (j = lo + 1; j <= lo + nx; j++) {
      irow[k] = j + nx;
      icol[k] = j;
      a[k] = -1.0/h2;
      k++;
    }
  }

  /* Set some options via iuser array and routine argument OPTION.
   * iuser[0] = print level, iuser[1] = iteration limit,
   * iuser[2]>0 means shifted-invert mode
   * iuser[3]>0 means print monitoring info.
   */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]", &prtlvl, &maxit);
#else
  scanf("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]", &prtlvl, &maxit);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]", &mode, &imon);
#else
  scanf("%"NAG_IFMT"%*[^\n]%"NAG_IFMT"%*[^\n]", &mode, &imon);
#endif

  if (imon>0) {
    /* Open the monitoring file for writing using
     * nag_open_file (x04acc).
     * If prtlvl >=10 internal monitoring information is also written.
     */
    fmode = 1;
```

```
    nag_open_file(filename, fmode, &fileid, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_open_file (x04acc) %s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    iuser[4] = fileid;
  }

  iuser[0] = prtlvl;
  iuser[1] = maxit;
  iuser[2] = mode;
  iuser[3] = imon;

  /* Compute eigenvalues and eigenvectors using
   * nag_eigen_real_symm_sparse_arnoldi (f02fkc).
   *   selected eigenvalues of real general matrix (driver).
   */
  nag_eigen_real_symm_sparse_arnoldi(n, nnz, a, irow, icol, nev, ncv, sigma,
                                     mymonit, myoption, &nconv, w, v, tdv,
                                     resid, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from  nag_eigen_real_symm_sparse_arnoldi (f02fkc)\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  if (imon>0) {
    /* Close the monitoring file using
     * nag_close_file (x04adc).
     */
    nag_close_file(fileid, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_close_file (x04adc) %s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }

  printf(" The %4"NAG_IFMT" ", nconv);
  printf(" Ritz values of closest to %13.5e are \n", sigma);
  for (i = 0; i < nconv; i++) {
    /* nag_machine_precision (x02ajc) */
    if (resid[i] > (double) (100 * n) * nag_machine_precision) {
      printf("%7"NAG_IFMT"  %13.5e %13.5e\n", i+1, w[i], resid[i]);
    } else {
      printf("%8"NAG_IFMT"  %13.5e\n", i+1, w[i]);
    }
  }

 END:

  NAG_FREE(w);
  NAG_FREE(a);
  NAG_FREE(v);
  NAG_FREE(resid);
  NAG_FREE(icol);
  NAG_FREE(irow);
  return exit_status;
}

static void NAG_CALL myoption(Integer icomm[], double com[], Integer *istat,
                              Nag_Comm *comm)
{
  NagError fail1;
  char     rec[26];

  INIT_FAIL(fail1);

  /* Set options using
```

```
   * nag_real_symm_sparse_eigensystem_option (f12fdc).
   */
  if (comm->iuser[0] > 0) {
#ifdef _WIN32
    sprintf_s(rec, _countof(rec), "Print Level=%5"NAG_IFMT, comm->iuser[0]);
#else
    sprintf(rec, "Print Level=%5"NAG_IFMT, comm->iuser[0]);
#endif
    fail1.code = 1;
    nag_real_symm_sparse_eigensystem_option(rec, icomm, com, &fail1);
    *istat = MAX(*istat, fail1.code);
  }

  if (comm->iuser[1] > 100) {
#ifdef _WIN32
    sprintf_s(rec, _countof(rec), "Iteration Limit=%5"NAG_IFMT, comm->iuser[1]);
#else
    sprintf(rec, "Iteration Limit=%5"NAG_IFMT, comm->iuser[1]);
#endif
    fail1.code = 1;
    nag_real_symm_sparse_eigensystem_option(rec, icomm, com, &fail1);
    *istat = MAX(*istat, fail1.code);
  }

  if (comm->iuser[2] > 0) {
    fail1.code = 1;
    nag_real_symm_sparse_eigensystem_option("Shifted Inverse", icomm, com,
                                            &fail1);
    *istat = MAX(*istat, fail1.code);
  }

  if (comm->iuser[3] > 0) {
    fail1.code = 1;
#ifdef _WIN32
    sprintf_s(rec, _countof(rec), "Monitoring=%5"NAG_IFMT, comm->iuser[4]);
#else
    sprintf(rec, "Monitoring=%5"NAG_IFMT, comm->iuser[4]);
#endif
    nag_real_symm_sparse_eigensystem_option(rec, icomm, com, &fail1);
    *istat = MAX(*istat, fail1.code);
  }
}

static void NAG_CALL mymonit(Integer ncv, Integer niter, Integer nconv,
                             const double w[], const double rzest[],
                             Integer *istat, Nag_Comm *comm)
{
  Integer i;
  char    line[100];

  if (comm->iuser[3] > 0) {

    /* Write lines to the file we opened for monitoring using
     * nag_write_line (x04bac).
     */

    if (niter == 1 && comm->iuser[2] > 0) {
#ifdef _WIN32
      sprintf_s(line, _countof(line), " Arnoldi basis vectors used: %4"NAG_IFMT
                "\n", ncv);
      nag_write_line(comm->iuser[4], line);
      sprintf_s(line, _countof(line), " The following Ritz values (mu) are "
                "related to the\n");
      nag_write_line(comm->iuser[4], line);
      sprintf_s(line, _countof(line), " true eigenvalues (lambda) by lambda = "
                "sigma + 1/mu\n");
#else
      sprintf(line, " Arnoldi basis vectors used: %4"NAG_IFMT"\n", ncv);
      nag_write_line(comm->iuser[4], line);
      sprintf(line, " The following Ritz values (mu) are related to the\n");
      nag_write_line(comm->iuser[4], line);
```

```
      sprintf(line, " true eigenvalues (lambda) by lambda = sigma + 1/mu\n");
#endif
      nag_write_line(comm->iuser[4], line);
    }

#ifdef _WIN32
    sprintf_s(line, _countof(line), "\n Iteration number %4"NAG_IFMT"\n",
              niter);
    nag_write_line(comm->iuser[4], line);
    sprintf_s(line, _countof(line), " Ritz values converged so far "
              "(%4"NAG_IFMT") and their Ritz estimates:\n", nconv);
#else
    sprintf(line, "\n Iteration number %4"NAG_IFMT"\n", niter);
    nag_write_line(comm->iuser[4], line);
    sprintf(line, " Ritz values converged so far (%4"NAG_IFMT") and their Ritz "
            "estimates:\n", nconv);
#endif
    nag_write_line(comm->iuser[4], line);

    for (i = 0; i < nconv; i++) {
#ifdef _WIN32
      sprintf_s(line, _countof(line), "  %4"NAG_IFMT" %13.5e %13.5e\n",
                i+1, w[i], rzest[i]);
#else
      sprintf(line, "  %4"NAG_IFMT" %13.5e %13.5e\n", i+1, w[i], rzest[i]);
#endif
      nag_write_line(comm->iuser[4], line);
    }

#ifdef _WIN32
    sprintf_s(line, _countof(line), " Next (unconverged) Ritz value:\n");
    nag_write_line(comm->iuser[4], line);
    sprintf_s(line, _countof(line), "  %4"NAG_IFMT" %13.5e\n", nconv + 1,
              w[nconv]);
#else
    sprintf(line, " Next (unconverged) Ritz value:\n");
    nag_write_line(comm->iuser[4], line);
    sprintf(line, "  %4"NAG_IFMT" %13.5e\n", nconv + 1, w[nconv]);
#endif
    nag_write_line(comm->iuser[4], line);
  }
  *istat = 0;
}
```

10.2 Program Data

```
nag_eigen_real_symm_sparse_arnoldi (f02fkc) Example Program Data
 20      : nx, matrix order n = nx*nx
 8       : nev, number of eigenvalues requested
 20      : ncv, size of subspace
 1.0     : sigma, shift (want eigenvalues close to sigma)
 0       : print level
 500     : maximum number of iterations
 1       : mode (0 = regular, 1 = shifted inverse)
 0       : imon (0 = no monitoring, 1 = monitoring on)
```

10.3 Program Results

```
nag_eigen_real_symm_sparse_arnoldi (f02fkc) Example Program Results

 The    8  Ritz values of closest to   1.00000e+00 are
       1    1.97024e+01
       2    4.90360e+01
       3    4.90360e+01
       4    7.83696e+01
       5    9.71967e+01
       6    9.71967e+01
       7    1.26530e+02
       8    1.26530e+02
```

11 Optional Arguments

Internally nag_eigen_real_symm_sparse_arnoldi (f02fkc) calls functions from the suite nag_real_symm_sparse_eigensystem_init (f12fac), nag_real_symm_sparse_eigensystem_iter (f12fbc), nag_real_symm_sparse_eigensystem_sol (f12fcc), nag_real_symm_sparse_eigensystem_option (f12fdc) and nag_real_symm_sparse_eigensystem_monit (f12fec). Several optional arguments for these computational functions define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_eigen_real_symm_sparse_arnoldi (f02fkc) these optional arguments are also used here and have associated *default values* that are usually appropriate. Therefore, you need only specify those optional arguments whose values are to be different from their default values.

Optional arguments may be specified via the user-supplied function **option** in the call to nag_eigen_real_symm_sparse_arnoldi (f02fkc). **option** must be coded such that one call to nag_real_symm_sparse_eigensystem_option (f12fdc) is necessary to set each optional argument. All optional arguments you do not specify are set to their default values.

The remainder of this section can be skipped if you wish to use the default values for all optional arguments.

The following is a list of the optional arguments available. A full description of each optional argument is provided in Section 11.1.

Advisory
Both Ends
Defaults
Iteration Limit
Largest Algebraic
Largest Magnitude
List
Monitoring
Nolist
Print Level
Regular
Regular Inverse
Shifted Inverse
Smallest Algebraic
Smallest Magnitude
Tolerance

11.1 Description of the Optional Arguments

For each option, we give a summary line, a description of the optional argument and details of constraints.

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value, where the symbol ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

Keywords and character values are case and white space insensitive.

Optional arguments used to specify files (e.g., **Advisory** and **Monitoring**) have type Integer. This Integer value corresponds with the Nag_FileID as returned by nag_open_file (x04acc). See Section 10 for an example of the use of this facility.

Advisory Default $= 0$

(See Section 3.2.1.1 in the Essential Introduction for further information on NAG data types.)

If the optional argument **List** is set then optional argument specifications are listed in a **List *file*** by setting the option to a file identification (unit) number associated with **Advisory** messages (see nag_open_file (x04acc)).

Defaults

This special keyword may be used to reset all optional arguments to their default values.

Iteration Limit i Default $= 300$

The limit on the number of Lanczos iterations that can be performed before nag_real_symm_sparse_eigensystem_iter (f12fbc) exits. If not all requested eigenvalues have converged to within **Tolerance** and the number of Lanczos iterations has reached this limit then nag_real_symm_sparse_eigensystem_iter (f12fbc) exits with an error; nag_real_symm_sparse_eigensystem_sol (f12fcc) can still be called subsequently to return the number of converged eigenvalues, the converged eigenvalues and, if requested, the corresponding eigenvectors.

Largest Magnitude Default
Both Ends
Largest Algebraic
Smallest Algebraic
Smallest Magnitude

The Lanczos iterative method converges on a number of eigenvalues with given properties. The default is for nag_real_symm_sparse_eigensystem_iter (f12fbc) to compute the eigenvalues of largest magnitude using **Largest Magnitude**. Alternatively, eigenvalues may be chosen which have **Largest Algebraic** part, **Smallest Magnitude**, or **Smallest Algebraic** part; or eigenvalues which are from **Both Ends** of the algebraic spectrum.

Nolist Default
List

Normally each optional argument specification is not listed as it is supplied. This behaviour can be changed using the **List** and **Nolist** options.

Monitoring Default $= -1$

(See Section 3.2.1.1 in the Essential Introduction for further information on NAG data types.)

Unless **Monitoring** is set to -1 (the default), monitoring information is output to Nag_FileID **Monitoring** during the solution of each problem; this may be the same as **Advisory**. The type of information produced is dependent on the value of **Print Level**, see the description of the optional argument **Print Level** in this section for details of the information produced. Please see nag_open_file (x04acc) to associate a file with a given Nag_FileID.

Print Level i Default $= 0$

This controls the amount of printing produced by nag_eigen_real_symm_sparse_arnoldi (f02fkc) as follows.

$= 0$ No output except error messages. If you want to suppress all output, set **Print Level** $= 0$.

> 0 The set of selected options.

$= 2$ Problem and timing statistics on final exit from nag_real_symm_sparse_eigensystem_iter (f12fbc).

≥ 5 A single line of summary output at each Lanczos iteration.

≥ 10 If **Monitoring** is set, then at each iteration, the length and additional steps of the current Lanczos factorization and the number of converged Ritz values; during re-orthogonalization, the norm of initial/restarted starting vector; on a final Lanczos iteration, the number of update iterations taken, the number of converged eigenvalues, the converged eigenvalues and their Ritz estimates.

≥ 20 Problem and timing statistics on final exit from nag_real_symm_sparse_eigensystem_iter (f12fbc). If **Monitoring** is set, then at each iteration, the number of shifts being applied, the eigenvalues and estimates of the symmetric tridiagonal matrix H, the size of the Lanczos basis, the wanted Ritz values and associated Ritz estimates and the shifts applied; vector norms prior to and following re-orthogonalization.

≥ 30 If **Monitoring** is set, then on final iteration, the norm of the residual; when computing the Schur form, the eigenvalues and Ritz estimates both before and after sorting; for each iteration, the norm of residual for compressed factorization and the symmetric tridiagonal matrix H; during re-orthogonalization, the initial/restarted starting vector; during the Lanczos iteration loop, a restart is flagged and the number of the residual requiring iterative refinement; while applying shifts, some indices.

≥ 40 If **Monitoring** is set, then during the Lanczos iteration loop, the Lanczos vector number and norm of the current residual; while applying shifts, key measures of progress and the order of H; while computing eigenvalues of H, the last rows of the Schur and eigenvector matrices; when computing implicit shifts, the eigenvalues and Ritz estimates of H.

≥ 50 If **Monitoring** is set, then during Lanczos iteration loop: norms of key components and the active column of H, norms of residuals during iterative refinement, the final symmetric tridiagonal matrix H; while applying shifts: number of shifts, shift values, block indices, updated tridiagonal matrix H; while computing eigenvalues of H: the diagonals of H, the computed eigenvalues and Ritz estimates.

Note that setting **Print Level** ≥ 30 can result in very lengthy **Monitoring** output.

Regular Default
Regular Inverse
Shifted Inverse

These options define the computational mode which in turn defines the form of operation OP(x) to be performed.

Regular OP $= A$
Shifted Inverse OP $= (A - \sigma I)^{-1}$ where σ is real
Regular Inverse OP $= A^{-1}$

Tolerance r Default $= \epsilon$

An approximate eigenvalue has deemed to have converged when the corresponding Ritz estimate is within **Tolerance** relative to the magnitude of the eigenvalue.