NAG Library Function Document

nag_real_symm_sparse_eigensystem_option (f12fdc)

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 this function need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 for a detailed description of the specification of the optional arguments*.

1 Purpose

nag_real_symm_sparse_eigensystem_option (f12fdc) is an option setting function in a suite of functions consisting of 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), and may be used to supply individual optional arguments to nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc). The initialization function nag_real_symm_sparse_eigensystem_init (f12fac) **must** have been called prior to calling nag_real_symm_sparse_eigensystem_option (f12fdc).

2 Specification

```
#include <nag.h>
#include <nagf12.h>
void nag_real_symm_sparse_eigensystem_option (const char *str,
     Integer icomm[], double comm[], NagError *fail)
```

3 Description

nag_real_symm_sparse_eigensystem_option (f12fdc) may be used to supply values for optional arguments to nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc). It is only necessary to call nag_real_symm_sparse_eigensystem_option (f12fdc) for those arguments whose values are to be different from their default values. One call to nag_real_symm_sparse_eigensystem_option (f12fdc) sets one argument value.

Each optional argument is defined by a single character string consisting of one or more items. The items associated with a given option must be separated by spaces, or equals signs [=]. Alphabetic characters may be upper or lower case. The string

```
'Iteration Limit = 500'
```

is an example of a string used to set an optional argument. For each option the string contains one or more of the following items:

– a mandatory keyword;

– a phrase that qualifies the keyword;

– a number that specifies an Integer or double value. Such numbers may be up to 16 contiguous characters in C's `d` or `g` format.

nag_real_symm_sparse_eigensystem_option (f12fdc) does not have an equivalent function from the ARPACK package which passes options by directly setting values to scalar arguments or to specific elements of array arguments. nag_real_symm_sparse_eigensystem_option (f12fdc) is intended to make the passing of options more transparent and follows the same principle as the single option setting functions in Chapter e04.

The setup function nag_real_symm_sparse_eigensystem_init (f12fac) must be called prior to the first call to nag_real_symm_sparse_eigensystem_option (f12fdc) and all calls to

nag_real_symm_sparse_eigensystem_option (f12fdc) must precede the first call to nag_real_symm_sparse_eigensystem_iter (f12fbc), the reverse communication iterative solver.

A complete list of optional arguments, their abbreviations, synonyms and default values is given in Section 11.

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: **str** – const char * *Input*

On entry: a single valid option string (as described in Section 3 and Section 11).

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

Note: the dimension, *dim*, of the array **icomm** must be at least $\max(1, \mathbf{licomm})$ (see nag_real_symm_sparse_eigensystem_init (f12fac)).

On initial entry: must remain unchanged following a call to the setup function nag_real_symm_sparse_eigensystem_init (f12fac).

On exit: contains data on the current options set.

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

Note: the dimension, *dim*, of the array **comm** must be at least 60.

On initial entry: must remain unchanged following a call to the setup function nag_real_symm_sparse_eigensystem_init (f12fac).

On exit: contains data on the current options set.

4: **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_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

Ambiguous keyword: ⟨*value*⟩

Keyword not recognized: ⟨*value*⟩

Second keyword not recognized: ⟨*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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example solves $Ax = \lambda Bx$ in **Shifted Inverse** mode, where A and B are 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. Data is passed to and from the reverse communication function nag_real_symm_sparse_eigensystem_iter (f12fbc) using pointers to the communication array.

10.1 Program Text

```
/* nag_real_symm_sparse_eigensystem_option (f12fdc) Example Program.
 *
 * Copyright 2014 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 mv(Integer, double *, double *);
static void my_dgttrf(Integer, double *, double *, double *,
                      double *, Integer *, Integer *);
static void my_dgttrs(Integer, double *, double *, double *,
                      double *, Integer *, double *, double *);

int main(void)
{
  /* Constants */
  Integer  licomm = 140, imon = 0;

  /* 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);

  printf("nag_real_symm_sparse_eigensystem_option (f12fdc) Example "
         "Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read values for nx, nev and cnv from data file. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &nev, &ncv);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &nev, &ncv);
#endif

  /* 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)))
    {
      printf("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);
  if (fail.code != NE_NOERROR)
    {
      printf(
        "Error from nag_real_symm_sparse_eigensystem_init (f12fac).\n%s\n",
        fail.message);
      exit_status = 1;
      goto END;
    }
  /* 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 using
     nag_real_symm_sparse_eigensystem_option (f12fdc). */
  nag_real_symm_sparse_eigensystem_option("shifted inverse", icomm, comm,
                                          &fail);
  if (fail.code != NE_NOERROR)
    {
```

```
      printf(
        "Error from nag_real_symm_sparse_eigensystem_option (f12fdc).\n%s\n",
        fail.message);
      exit_status = 1;
      goto END;
    }

  /* Setup M and factorise */
  h = 1.0 / (double)(n + 1);
  r1 = 2.0 * h / 3.0;
  r2 = h / 6.0;
  sigma = 0.0;
  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 <--- OP*x = inv[A-SIGMA*M]*M*x. */
          mv(n, x, x2);
          my_dgttrs(n, dl, dd, du, du2, ipiv, x2, y);
        }
      else if (irevcm == 1)
        {
          /* Perform y <-- OP*x = inv[A-sigma*M]*M*x;
             M*x has been saved in COMM(ICOMM(3)) or MX. */
          my_dgttrs(n, dl, dd, du, du2, ipiv, mx, y);
        }
      else if (irevcm == 2)
        {
          /* Perform  y <--- M*x. */
          mv(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);
          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_symm_sparse_eigensystem_sol
         (f12fcc) to compute eigenvalues/vectors. */
      nag_real_symm_sparse_eigensystem_sol(&nconv, eigv, v, sigma, resid, v,
                                           comm, icomm, &fail);
      printf("\n The %4"NAG_IFMT" generalized Ritz values", nconv);
      printf(" closest to %8.4f are:\n\n", sigma);
      for (j = 0; j <= nconv-1; ++j)
        {
```

```
          printf("%8"NAG_IFMT"%5s%12.4f\n", j+1, "", eigv[j]);
        }
    }
  else
    {
      printf(" Error from nag_real_symm_sparse_eigensystem_iter "
              "(f12fbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(dd);
  NAG_FREE(dl);
  NAG_FREE(du);
  NAG_FREE(du2);
  NAG_FREE(comm);
  NAG_FREE(eigv);
  NAG_FREE(eigest);
  NAG_FREE(resid);
  NAG_FREE(v);
  NAG_FREE(x2);
  NAG_FREE(icomm);
  NAG_FREE(ipiv);

  return exit_status;
}

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

  /* Function Body */
  h = 1.0 / ((double)(n + 1) * 6.);
  y[0] = h*(v[0] * 4.0 + v[1]);
  for (j = 1; j <= n - 2; ++j)
    {
      y[j] = h*(v[j-1] + v[j] * 4.0 + v[j+1]);
    }
  y[n-1] = h*(v[n-2] + v[n-1] * 4.0);
  return;
} /* mv */

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;
}
```

10.2 Program Data

```
nag_real_symm_sparse_eigensystem_option (f12fdc) Example Program Data
 100  4  10 : Values for n, nev and ncv
```

10.3 Program Results

```
nag_real_symm_sparse_eigensystem_option (f12fdc) Example Program Results

 The    4 generalized Ritz values closest to   0.0000 are:

       1           9.8704
       2          39.4912
       3          88.8909
       4         158.1175
```

11 Optional Arguments

Several optional arguments for the computational functions nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc) these optional arguments have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional arguments whose values are to be different from 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
Buckling
Cayley
Defaults
Exact Shifts
Generalized
Initial Residual
Iteration Limit
Largest Algebraic
Largest Magnitude
List
Monitoring

Nolist
Print Level
Random Residual
Regular
Regular Inverse
Shifted Inverse
Smallest Algebraic
Smallest Magnitude
Standard
Supplied Shifts
Tolerance
Vectors

Optional arguments may be specified by calling nag_real_symm_sparse_eigensystem_option (f12fdc) before a call to nag_real_symm_sparse_eigensystem_iter (f12fbc), but after a call to nag_real_symm_sparse_eigensystem_init (f12fac). One call is necessary for each optional argument.

All optional arguments you do not specify are set to their default values. Optional arguments you do specify are unaltered by nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc) (unless they define invalid values) and so remain in effect for subsequent calls unless you alter them.

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 Nag_FileID. This ID value must either be set to 0 (the default value) in which case there will be no output, or will be as returned by a call of nag_open_file (x04acc).

Advisory Default $= 0$

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

Advisory messages are output to Nag_FileID **Advisory** during the solution of the problem.

Defaults

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

Exact Shifts Default
Supplied Shifts

During the Lanczos iterative process, shifts are applied internally as part of the implicit restarting scheme. The shift strategy used by default and selected by the **Exact Shifts** is strongly recommended over the alternative **Supplied Shifts** (see Lehoucq *et al.* (1998) for details of shift strategies).

If **Exact Shifts** are used then these are computed internally by the algorithm in the implicit restarting scheme.

If **Supplied Shifts** are used then, during the Lanczos iterative process, you must supply shifts through array arguments of nag_real_symm_sparse_eigensystem_iter (f12fbc) when nag_real_symm_sparse_eigensystem_iter (f12fbc) returns with **irevcm** $= 3$; the real and imaginary parts of the shifts are supplied in **y** and **mx** respectively. This option should only be used if you are an experienced user since this requires some algorithmic knowledge and because more operations are usually required than for the implicit shift scheme. Details on the use of explicit shifts and further references on shift strategies are available in Lehoucq *et al.* (1998).

Iteration Limit i Default $= 300$

The limit on the number of Lanczositerations 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 Lanczositerations 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.

Note that these options select the eigenvalue properties for eigenvalues of OP (and B for **Generalized** problems), the linear operator determined by the computational mode and problem type.

Nolist Default
List

Normally each optional argument specification is not printed to **Advisory** as it is supplied. Optional argument **List** may be used to enable printing and optional argument **Nolist** may be used to suppress the printing.

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_real_symm_sparse_eigensystem_option (f12fdc) 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.

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

Random Residual Default
Initial Residual

To begin the Lanczos iterative process, nag_real_symm_sparse_eigensystem_iter (f12fbc) requires an initial residual vector. By default nag_real_symm_sparse_eigensystem_iter (f12fbc) provides its own random initial residual vector; this option can also be set using optional argument **Random Residual**. Alternatively, you can supply an initial residual vector (perhaps from a previous computation) to nag_real_symm_sparse_eigensystem_iter (f12fbc) through the array argument **resid**; this option can be set using optional argument **Initial Residual**.

Regular Default
Regular Inverse
Shifted Inverse
Buckling
Cayley

These options define the computational mode which in turn defines the form of operation $\mathrm{OP}(x)$ to be performed when nag_real_symm_sparse_eigensystem_iter (f12fbc) returns with **irevcm** $= -1$ or 1 and the matrix-vector product Bx when nag_real_symm_sparse_eigensystem_iter (f12fbc) returns with **irevcm** $= 2$.

Given a **Standard** eigenvalue problem in the form $Ax = \lambda x$ then the following modes are available with the appropriate operator $\mathrm{OP}(x)$.

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

Given a **Generalized** eigenvalue problem in the form $Ax = \lambda Bx$ then the following modes are available with the appropriate operator OP(x).

Regular Inverse	OP $= B^{-1}A$
Shifted Inverse	OP $= (A - \sigma B)^{-1}B$, where σ is real
Buckling	OP $= (B - \sigma A)^{-1}A$, where σ is real
Cayley	OP $= (A - \sigma B)^{-1}(A + \sigma B)$, where σ is real

Standard Default
Generalized

The problem to be solved is either a standard eigenvalue problem, $Ax = \lambda x$, or a generalized eigenvalue problem, $Ax = \lambda Bx$. The optional argument **Standard** should be used when a standard eigenvalue problem is being solved and the optional argument **Generalized** should be used when a generalized eigenvalue problem is being solved.

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.

Vectors Default = RITZ

The function nag_real_symm_sparse_eigensystem_sol (f12fcc) can optionally compute the Schur vectors and/or the eigenvectors corresponding to the converged eigenvalues. To turn off computation of any vectors the option **Vectors** = NONE should be set. To compute only the Schur vectors (at very little extra cost), the option **Vectors** = SCHUR should be set and these will be returned in the array argument **v** of nag_real_symm_sparse_eigensystem_sol (f12fcc). To compute the eigenvectors (Ritz vectors) corresponding to the eigenvalue estimates, the option **Vectors** = RITZ should be set and these will be returned in the array argument **z** of nag_real_symm_sparse_eigensystem_sol (f12fcc), if **z** is set equal to **v** (as in Section 10) then the Schur vectors in **v** are overwritten by the eigenvectors computed by nag_real_symm_sparse_eigensystem_sol (f12fcc).