

NAG C Library Function Document

nag_real_symm_sparse_eigensystem_option (f12fdc)

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_option (f12fdc), 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_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 10.

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: **str** – const char * *Input*
On entry: a single valid option string (as described in Section 3 above and in Section 10).
- 2: **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)).
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 $\max(1, \mathbf{lcomm})$ (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.
- 4: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ 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 consult NAG for assistance.

NE_INVALID_OPTION

Ambiguous keyword: $\langle value \rangle$

Keyword not recognized: $\langle value \rangle$

Second keyword not recognized: $\langle value \rangle$

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

The 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 $\frac{\partial^2 y}{\partial x^2}$ 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.

9.1 Program Text

```

/* nag_real_symm_sparse_eigensystem_option (f12fdc) 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 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);

    Vprintf("nag_real_symm_sparse_eigensystem_option (f12fdc) 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 using
nag_real_symm_sparse_eigensystem_option (f12fdc). */
nag_real_symm_sparse_eigensystem_option("shifted inverse",
icomm, comm, &fail);

/* 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_dgtrf(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);
    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\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 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 dgtrf 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_option (f12fdc) Example Program Data
 100 4 10 : Values for n, nev and ncv

9.3 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

10 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. A complete list of optional arguments and their default values is given in Section 10.1.

Optional arguments may be specified by calling `nag_real_symm_sparse_eigensystem_option` (f12fdc) prior to 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 not specified by you are set to their default values. Optional arguments specified by you 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 altered by you.

10.1 Optional Argument Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 10.2. The minimum abbreviation of each keyword is underlined. The qualifier may be omitted. The letters *i* and *r* denote Integer and double values required with certain options. The number ϵ is a generic notation for *machine precision* (see `nag_machine_precision` (X02AJC)).

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).

Optional Arguments	Default Values
<u>Advisory</u>	Default = 0
<u>Both Ends</u>	
<u>Buckling</u>	
<u>Cayley</u>	
<u>Defaults</u>	
<u>Exact Shifts</u>	Default = Exact Shifts
<u>Generalized</u>	
<u>Initial Residual</u>	
<u>Iteration Limit</u>	Default = 300
<u>Largest Algebraic</u>	
<u>Largest Magnitude</u>	Default = Largest Magnitude
<u>List</u>	Default = Nolist
<u>Monitoring</u>	Default = -1
<u>Nolist</u>	
<u>Print Level</u>	Default = 0
<u>Random Residual</u>	Default = Random Residual
<u>Regular</u>	Default = Regular
<u>Regular Inverse</u>	
<u>Shifted Inverse</u>	
<u>Smallest Algebraic</u>	
<u>Smallest Magnitude</u>	
<u>Standard</u>	Default = Standard
<u>Supplied Shifts</u>	
<u>Tolerance</u>	Default = ϵ
<u>Vectors</u>	Default = Schur

10.2 Description of the Optional Arguments

Advisory – Nag_FileID Default = 0

(See Section 10.1 for a description of **Nag_FileID**.)

Advisory messages are output to **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
Supplied Shifts

Default = **Exact 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 option **Exact Shifts** is strongly recommended over the alternative option **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)`; 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.

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 `irevcn = 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 – Integer

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
Largest Algebraic
Smallest Magnitude
Smallest Algebraic
Both Ends

Default = **Largest 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 option **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.

List
Nolist

Default = **Nolist**

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

Monitoring – Nag_FileID

Default = -1

(See Section 10.1 for a description of **Nag_FileID**.)

Unless **Monitoring** is set to -1 (the default), monitoring information is output to **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 **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 – Integer *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-orthogonalisation, 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-orthogonalisation.
- ≥ 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-orthogonalisation, 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.

Random Residual Default = **Random Residual**
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 **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 **Random Residual**.

Regular Default = **Regular**
Regular Inverse
Shifted Inverse
Buckling
Cayley

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

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

