NAG C Library Function Document

nag_real_sparse_eigensystem_option (f12adc)

1 Purpose

nag_real_sparse_eigensystem_option (f12adc) is an option setting function in a suite of functions consisting of nag_real_sparse_eigensystem_option (f12adc), nag_real_sparse_eigensystem_init (f12aac), nag_real_sparse_eigensystem_iter (f12abc), nag_real_sparse_eigensystem_sol (f12acc) and nag_real_sparse_eigensystem_monit (f12aec), and may be used to supply individual optional arguments to nag_real_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc). The initialization function nag_real_sparse_eigensystem_init (f12aac) **must** have been called prior to calling nag_real_sparse_eigensystem_option (f12adc).

2 Specification

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

void nag_real_sparse_eigensystem_option (const char *str, Integer icomm[],
     double comm[], NagError *fail)
```

3 Description

nag_real_sparse_eigensystem_option (f12adc) may be used to supply values for optional arguments to nag_real_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc). It is only necessary to call nag_real_sparse_eigensystem_option (f12adc) for those arguments whose values are to be different from their default values. One call to nag_real_sparse_eigensystem_option (f12adc) 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

```
Vectors = None
```

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_sparse_eigensystem_option (f12adc) 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_sparse_eigensystem_option (f12adc) 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_sparse_eigensystem_init (f12aac) must be called prior to the first call to nag_real_sparse_eigensystem_option (f12adc) and all calls to nag_real_sparse_eigensystem_option (f12adc) must precede the first call to nag_real_sparse_eigensystem_iter (f12abc), 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, Philidelphia

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 $\max(1, \mathbf{licomm})$ (see nag_real_sparse_eigensystem_init (f12aac)).

On initial entry: must remain unchanged following a call to the setup function nag_real_sparse_eigensystem_init (f12aac).

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_sparse_eigensystem_init (f12aac)).

On initial entry: must remain unchanged following a call to the setup function nag_real_sparse_eigensystem_init (f12aac).

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 ⟨*value*⟩ had an illegal value.

NE_INITIALIZATION

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

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: ⟨*value*⟩

Keyword not recognized: ⟨*value*⟩

Second keyword not recognized: ⟨*value*⟩

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 derived from the finite element discretization of the one-dimensional convection-diffusion operator $\frac{d^2u}{dx^2} + \rho\frac{du}{dx}$ on the interval $[0, 1]$, with zero Dirichlet boundary conditions.

The shift σ is a real number, and the operator used in the shifted-inverse iterative process is $\mathrm{OP} = (A - \sigma B)^{-1}B$.

9.1 Program Text

```
/* nag_real_sparse_eigensystem_option (f12adc) 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, rho, s, s1, s2, s3, sigmai, sigmar;
  Integer exit_status, info, irevcm, j, lcomm, n, nconv, ncv;
  Integer nev, niter, nshift, nx;
  /* Nag types */
  NagError fail;

  /* Arrays */
  double *comm=0, *dd=0, *dl=0, *du=0, *du2=0, *eigvr=0, *eigvi=0;
  double *eigest=0, *resid=0, *x2=0, *v=0;
  Integer *icomm=0, *ipiv=0;
  /* Pointers */
  double *mx=0, *x=0, *y=0;

  exit_status = 0;
  INIT_FAIL(fail);

  Vprintf("nag_real_sparse_eigensystem_option (f12adc) Example Program Results"
          "\n");
  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  /* Read problem parameter values from data file. */
  Vscanf("%ld%ld%ld%lf%lf%lf%*[^\n] ", &nx, &nev, &ncv,
         &rho, &sigmar, &sigmai);
```

```
  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)) ||
       !(dd = NAG_ALLOC(n, double)) ||
       !(dl = NAG_ALLOC(n, double)) ||
       !(du = NAG_ALLOC(n, double)) ||
       !(du2 = NAG_ALLOC(n, 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_sparse_eigensystem_init (f12aac). */
  nag_real_sparse_eigensystem_init(n, nev, ncv, icomm, licomm, comm,
                                   lcomm, &fail);
  /* Select the required spectrum using
     nag_real_sparse_eigensystem_option (f12adc). */
  nag_real_sparse_eigensystem_option("SHIFTED REAL", icomm, comm, &fail);
  /* Select the problem type using
     nag_real_sparse_eigensystem_option (f12adc). */
  nag_real_sparse_eigensystem_option("GENERALIZED", icomm, comm, &fail);
  /* Construct C = A - SIGMA*I, and factor C using my_dgttrf. */
  h = 1.0 / (double) (n + 1);
  s = rho / 2.0;
  s1 = -1.0 / h - s - sigmar * h / 6.0;
  s2 = 2.0 / h - sigmar * 4.0 * h / 6.0;
  s3 = -1.0 / h + s - sigmar * h / 6.0;
  for (j = 0; j <= n - 2; ++j)
    {
      dl[j] = s1;
      dd[j] = s2;
      du[j] = s3;
    }
  dd[n - 1] = s2;

  my_dgttrf(n, dl, dd, du, du2, ipiv, &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)
        {
          /* Perform  y <--- OP*x = inv[A-SIGMA*M]*M*x using
             my_dggtrs */
          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 where
             mx is available. */
          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_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);
          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_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. */

      Vprintf("\n The %4ld generalized Ritz values closest", nconv);
      Vprintf(" to  unity are:\n\n");
      for (j = 0; j <= nconv-1; ++j)
        {
          Vprintf("%8ld%5s( %12.4f ,%12.4f )\n", j+1,  "",
                  sigmar + 1.0/eigvr[j], eigvi[j]);
        }
    }
  else
    {
      Vprintf(" Error from nag_real_sparse_eigensystem_iter (f12abc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (comm) NAG_FREE(comm);
  if (eigvr) NAG_FREE(eigvr);
  if (eigvi) NAG_FREE(eigvi);
  if (eigest) NAG_FREE(eigest);
  if (dd) NAG_FREE(dd);
  if (dl) NAG_FREE(dl);
  if (du) NAG_FREE(du);
  if (du2) NAG_FREE(du2);
  if (resid) NAG_FREE(resid);
  if (v) NAG_FREE(v);
  if (icomm) NAG_FREE(icomm);
  if (ipiv) NAG_FREE(ipiv);
  return exit_status;
}

static void mv(Integer n, double *v, double *y)
{
  /* Compute the matrix vector multiplication Y<---M*X, where M is
     mass matrix formed by using piecewise linear elements on [0,1]. */

  /* Scalars */
  double h;
  Integer j;

  /* Function Body */
  h = 1.0 / (double) (6*(n + 1));
  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;
}
```

9.2 Program Data

```
nag_real_sparse_eigensystem_option (f12adc) Example Program Data
 10 4 10 10.0 1.0 0.0 : Values for nx, nev, ncv, rho, sigmar, sigmai
```

9.3 Program Results

```
nag_real_sparse_eigensystem_option (f12adc) Example Program Results

 The    4 generalized Ritz values closest to  unity are:

       1     (       1.0287 ,      0.0000 )
       2     (       1.0155 ,      0.0000 )
       3     (       1.0088 ,      0.0000 )
       4     (       1.0055 ,      0.0000 )
```

10 Optional Arguments

Several optional arguments for the computational functions nag_real_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_real_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc) 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_sparse_eigensystem_option (f12adc) prior to a call to nag_real_sparse_eigensystem_iter (f12abc), but after a call to nag_real_sparse_eigensystem_init (f12aac). 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_sparse_eigensystem_iter (f12abc) and nag_real_sparse_eigensystem_sol (f12acc) (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**
Advisory	Default $= 0$
Defaults	
Exact Shifts	Default $=$ **Exact Shifts**
Generalized	
Initial Residual	
Iteration Limit	Default $= 300$
Largest Imaginary	
Largest Magnitude	Default $=$ **Largest Magnitude**
Largest Real	
List	Default $=$ **Nolist**
Monitoring	Default $= -1$
Nolist	
Print Level	Default $= 0$
Random Residual	Default $=$ **Random Residual**
Regular	Default $=$ **Regular**
Regular Inverse	
Shifted Inverse Imaginary	
Shifted Inverse Real	
Smallest Imaginary	
Smallest Magnitude	
Smallest Real	
Standard	Default $=$ **Standard**
Supplied Shifts	
Tolerance	Default $= \epsilon$
Vectors	Default $=$ Ritz

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 Default $=$ **Exact Shifts**
Supplied Shifts

During the Arnoldi 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 Arnoldi iterative process, you must supply shifts through array arguments of nag_real_sparse_eigensystem_iter (f12abc) when nag_real_sparse_eigensystem_iter (f12abc) 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 – Integer i Default $= 300$

The limit on the number of Arnoldi iterations that can be performed before nag_real_sparse_eigensystem_iter (f12abc) exits. If not all requested eigenvalues have converged to within **Tolerance** and the number of Arnoldi iterations has reached this limit then nag_real_sparse_eigensystem_iter (f12abc) exits with an error; nag_real_sparse_eigensystem_sol (f12acc) can still be called subsequently to return the number of converged eigenvalues, the converged eigenvalues and, if requested, the corresponding eigenvectors.

Largest Magnitude Default $=$ **Largest Magnitude**
Largest Real
Largest Imaginary
Smallest Magnitude
Smallest Real
Smallest Imaginary

The Arnoldi iterative method converges on a number of eigenvalues with given properties. The default is for nag_real_sparse_eigensystem_iter (f12abc) to compute the eigenvalues of largest magnitude using option **Largest Magnitude**. Alternatively, eigenvalues may be chosen which have **Largest Real** part, **Largest Imaginary** part, **Smallest Magnitude**, **Smallest Real** part or **Smallest Imaginary** part.

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 Default $=$ **Nolist**
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_sparse_eigensystem_option (f12adc) 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_sparse_eigensystem_iter (f12abc).

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

≥ 10 If **Monitoring** is set, then at each iteration, the length and additional steps of the current Arnoldi factorization and the number of converged Ritz values; during re-orthogonalisation, the norm of initial/restarted starting vector.

≥ 20 Problem and timing statistics on final exit from nag_real_sparse_eigensystem_iter (f12abc). If **Monitoring** is set, then at each iteration, the number of shifts being applied, the eigenvalues and estimates of the Hessenberg matrix H, the size of the Arnoldi 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 real Schur form, the eigenvalues and Ritz estimates both before and after sorting; for each iteration, the norm of residual for compressed factorization and the compressed upper Hessenberg matrix H; during re-orthogonalisation, the initial/restarted starting vector; during the Arnoldi 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 Arnoldi iteration loop, the Arnoldi 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 Arnoldi iteration loop: norms of key components and the active column of H, norms of residuals during iterative refinement, the final upper Hessenberg matrix H; while applying shifts: number of shifts, shift values, block indices, updated matrix H; while computing eigenvalues of H: the matrix H, the computed eigenvalues and Ritz estimates.

Random Residual Default $=$ **Random Residual**
Initial Residual

To begin the Arnoldi iterative process, nag_real_sparse_eigensystem_iter (f12abc) requires an initial residual vector. By default nag_real_sparse_eigensystem_iter (f12abc) 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_sparse_eigensystem_iter (f12abc) through the array argument **resid**; this option can be set using **Random Residual**.

Regular Default $=$ **Regular**
Regular Inverse
Shifted Inverse Real
Shifted Inverse Imaginary

These options define the computational mode which in turn defines the form of operation $\mathrm{OP}(x)$ to be performed when nag_real_sparse_eigensystem_iter (f12abc) returns with **irevcm** $= -1$ or 1 and the matrix-vector product Bx when nag_real_sparse_eigensystem_iter (f12abc) 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 Real	$\mathrm{OP} = (A - \sigma I)^{-1}$ 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	$\text{OP} = B^{-1}A$
Shifted Real with real shift	$\text{OP} = (A - \sigma B)^{-1}B$, where σ is real
Shifted Real with complex shift	$\text{OP} = \text{Real}\left((A - \sigma B)^{-1}B\right)$, where σ is complex
Shifted Imaginary	$\text{OP} = \text{Imag}\left((A - \sigma B)^{-1}B\right)$, where σ is complex

Standard Default = **Standard**
Generalized

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

Tolerance – double r Default = ϵ

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_sparse_eigensystem_sol (f12acc) 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_sparse_eigensystem_sol (f12acc). 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_sparse_eigensystem_sol (f12acc), if **z** is set equal to **v** (as in Section 9) then the Schur vectors in **v** are overwritten by the eigenvectors computed by nag_real_sparse_eigensystem_sol (f12acc).