

## NAG C Library Function Document

### nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc)

#### 1 Purpose

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) is the main solver function in a suite of functions consisting of nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc), nag\_real\_sparse\_eigensystem\_option (f12adc) and nag\_real\_banded\_sparse\_eigensystem\_init (f12afc), that must be called following an initial call to nag\_real\_banded\_sparse\_eigensystem\_init (f12afc) and following any calls to nag\_real\_sparse\_eigensystem\_option (f12adc).

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) returns approximations to selected eigenvalues, and (optionally) the corresponding eigenvectors, of a standard or generalized eigenvalue problem defined by real banded non-symmetric matrices. The banded matrix must be stored using the LAPACK column ordered storage format for real banded non-symmetric (see Section 3.3.2 in the f07 Chapter Introduction).

#### 2 Specification

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

```
void nag_real_banded_sparse_eigensystem_sol (Integer kl, Integer ku,
      const double ab[], const double mb[], double sigmar, double sigmai,
      Integer *nconv, double dr[], double di[], double z[], double resid[],
      double v[], double comm[], Integer icomm[], NagError *fail)
```

#### 3 Description

The suite of functions is designed to calculate some of the eigenvalues,  $\lambda$ , (and optionally the corresponding eigenvectors,  $x$ ) of a standard eigenvalue problem  $Ax = \lambda x$ , or of a generalized eigenvalue problem  $Ax = \lambda Bx$  of order  $n$ , where  $n$  is large and the coefficient matrices  $A$  and  $B$  are banded, real and non-symmetric.

Following a call to the initialization function nag\_real\_banded\_sparse\_eigensystem\_init (f12afc), nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) returns the converged approximations to eigenvalues and (optionally) the corresponding approximate eigenvectors and/or an orthonormal basis for the associated approximate invariant subspace. The eigenvalues (and eigenvectors) are selected from those of a standard or generalized eigenvalue problem defined by real banded non-symmetric matrices. There is negligible additional computational cost to obtain eigenvectors; an orthonormal basis is always computed, but there is an additional storage cost if both are requested.

The banded matrices  $A$  and  $B$  must be stored using the LAPACK column ordered storage format for banded non-symmetric matrices; please refer to Section 3.3.2 in the f07 Chapter Introduction for details on this storage format.

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) is based on the banded driver functions **dnbdr1** to **dnbdr6** from the ARPACK package, which uses the Implicitly Restarted Arnoldi iteration method. The method is described in Lehoucq and Sorensen (1996) and Lehoucq (2001) while its use within the ARPACK software is described in great detail in Lehoucq *et al.* (1998). An evaluation of software for computing eigenvalues of sparse non-symmetric matrices is provided in Lehoucq and Scott (1996). This suite of functions offers the same functionality as the ARPACK banded driver software for real non-symmetric problems, but the interface design is quite different in order to make the option setting clearer to you and to combine the different drivers into a general purpose function.

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc), is a general purpose forward communication function that must be called following initialization by nag\_real\_banded\_sparse\_eigensystem\_init (f12afc). nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) uses options, set either by default or explicitly by calling nag\_real\_sparse\_eigensystem\_option (f12adc), to return the converged approximations to selected eigenvalues and (optionally):

- the corresponding approximate eigenvectors;
- an orthonormal basis for the associated approximate invariant subspace;
- both.

## 4 References

Lehoucq R B (2001) Implicitly Restarted Arnoldi Methods and Subspace Iteration *SIAM Journal on Matrix Analysis and Applications* **23** 551–562

Lehoucq R B and Scott J A (1996) An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices *Preprint MCS-P547-1195* Argonne National Laboratory

Lehoucq R B and Sorensen D C (1996) Deflation Techniques for an Implicitly Restarted Arnoldi Iteration *SIAM Journal on Matrix Analysis and Applications* **17** 789–821

Lehoucq R B, Sorensen D C and Yang C (1998) *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philadelphia

## 5 Arguments

- 1: **kl** – Integer *Input*  
*On entry:* the number of subdiagonals of the matrices *A* and *B*.  
*Constraint:* **kl** ≥ 0.
- 2: **ku** – Integer *Input*  
*On entry:* the number of superdiagonals of the matrices *A* and *B*.  
*Constraint:* **ku** ≥ 0.
- 3: **ab**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **ab** must be at least  $\max(1, \mathbf{n} \times (2\mathbf{kl} + \mathbf{ku} + 1))$  (see `nag_real_banded_sparse_eigensystem_init (f12afc)`).  
*On entry:* must contain the matrix *A* in LAPACK column-ordered banded storage format for non-symmetric matrices (see Section 3.3.2 in the f07 Chapter Introduction).
- 4: **mb**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **mb** must be at least  $\max(1, \mathbf{n} \times (2\mathbf{kl} + \mathbf{ku} + 1))$  (see `nag_real_banded_sparse_eigensystem_init (f12afc)`).  
*On entry:* must contain the matrix *B* in LAPACK column-ordered banded storage format for non-symmetric matrices (see Section 3.3.2 in the f07 Chapter Introduction).
- 5: **sigmar** – double *Input*  
*On entry:* if one of the **Shifted** modes (see `nag_real_sparse_eigensystem_option (f12adc)`) have been selected then **sigmar** contains the real part of the shift used; otherwise **sigmar** is not referenced.
- 6: **sigmai** – double *Input*  
*On entry:* if one of the **Shifted** modes (see `nag_real_sparse_eigensystem_option (f12adc)`) have been selected then **sigmai** contains the imaginary part of the shift used; otherwise **sigmai** is not referenced.
- 7: **nconv** – Integer \* *Output*  
*On exit:* the number of converged eigenvalues.

- 8: **dr**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **dr** must be at least **nev** (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On exit:* the first **nconv** locations of the array **dr** contain the real parts of the converged approximate eigenvalues.
- 9: **di**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **di** must be at least **nev** (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On exit:* the first **nconv** locations of the array **di** contain the imaginary parts of the converged approximate eigenvalues.
- 10: **z**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **z** must be at least  $\mathbf{n} \times (\mathbf{nev} + 1)$  (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On exit:* if the default option **Vectors** = Ritz (see `nag_real_sparse_eigensystem_option` (f12adc)) has been selected then **z** contains the final set of eigenvectors corresponding to the eigenvalues held in **dr** and **di**. The complex eigenvector associated with the eigenvalue with positive imaginary part is stored in two consecutive array segments. The first segment holds the real part of the eigenvector and the second holds the imaginary part. The eigenvector associated with the eigenvalue with negative imaginary part is simply the complex conjugate of the eigenvector associated with the positive imaginary part.  
For example, if **di**[0] is non-zero, the first eigenvector has real parts stored in locations **z**[*i*], for  $i = 0, 1, \dots, \mathbf{n} - 1$  and imaginary parts stored in **z**[*i*], for  $i = \mathbf{n}, \mathbf{n} + 1, \dots, 2\mathbf{n} - 1$ .
- 11: **resid**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **resid** must be at least **n** (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On entry:* need not be set unless the option **Initial Residual** has been set in a prior call to `nag_real_sparse_eigensystem_option` (f12adc) in which case **resid** should contain an initial residual vector.  
*On exit:* contains the final residual vector.
- 12: **v**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **v** must be at least  $\max(1, \mathbf{n} \times \mathbf{ncv})$  (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On exit:* if the option **Vectors** has been set to Schur or Ritz and **z** is not set to **v**, then the first **nconv** sections of **v**, of length *n*, will contain approximate Schur vectors that span the desired invariant subspace. The *i*th Schur vector is stored in locations **v**[ $i \times \mathbf{n} + j$ ], for  $i = 0, 1, \dots, \mathbf{nconv} - 1$  and  $j = 0, 1, \dots, \mathbf{n} - 1$ .
- 13: **comm**[*dim*] – double *Communication Array*  
**Note:** the dimension, *dim*, of the array **comm** must be at least  $\max(1, \mathbf{lcomm})$  (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On initial entry:* must remain unchanged from the prior call to `nag_real_banded_sparse_eigensystem_init` (f12afc) and `nag_real_sparse_eigensystem_option` (f12adc).  
*On exit:* contains no useful information.

- 14: **icomm**[*dim*] – Integer *Communication Array*  
**Note:** the dimension, *dim*, of the array **icomm** must be at least  $\max(1, \mathbf{icomm})$  (see `nag_real_banded_sparse_eigensystem_init` (f12afc)).  
*On initial entry:* must remain unchanged from the prior call to `nag_real_banded_sparse_eigensystem_init` (f12afc) and `nag_real_sparse_eigensystem_option` (f12adc).  
*On exit:* contains no useful information.
- 15: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.6 of the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Error: unable to allocate requested internal workspace.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_COMP\_BAND\_FAC

Failure during internal factorization of complex banded matrix. Please contact NAG.

### NE\_COMP\_BAND\_SOL

Failure during internal solution of complex banded matrix. Please contact NAG.

### NE\_INITIALIZATION

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

### NE\_INT

The maximum number of iterations  $\leq 0$ , the option **Iteration Limit** has been set to  $\langle value \rangle$ .

On entry, **kl** =  $\langle value \rangle$ .

Constraint: **kl**  $\geq 0$ .

On entry, **ku** =  $\langle value \rangle$ .

Constraint: **ku**  $\geq 0$ .

### NE\_INTERNAL\_EIGVAL\_FAIL

Error in internal call to compute eigenvalues and corresponding error bounds of the current upper Hessenberg matrix. Please contact NAG.

### NE\_INTERNAL\_EIGVEC\_FAIL

Error in internal call to compute eigenvectors. Please contact NAG.

### NE\_INTERNAL\_ERROR

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

### NE\_INVALID\_OPTION

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

**NE\_MAX\_ITER**

The maximum number of iterations has been reached. The maximum number of iterations =  $\langle value \rangle$ . The number of converged eigenvalues =  $\langle value \rangle$ .

**NE\_NO\_ARNOLDI\_FAC**

Could not build an Arnoldi factorization. The size of the current Arnoldi factorization =  $\langle value \rangle$ .

**NE\_NO\_SHIFTS\_APPLIED**

No shifts could be applied during a cycle of the implicitly restarted Lanczos iteration.

**NE\_OPT\_INCOMPAT**

The options **Generalized** and **Regular** are incompatible.

**NE\_REAL\_BAND\_FAC**

Failure during internal factorization of real banded matrix. Please contact NAG.

**NE\_REAL\_BAND\_SOL**

Failure during internal solution of real banded matrix. Please contact NAG.

**NE\_SCHUR\_EIG\_FAIL**

During calculation of a real Schur form, there was a failure to compute a number of eigenvalues. Please contact NAG.

**NE\_SCHUR\_REORDER**

The computed Schur form could not be reordered by an internal call. Please contact NAG.

**NE\_TRANSFORM\_OVFL**

Overflow occurred during transformation of Ritz values to those of the original problem.

**NE\_ZERO\_EIGS\_FOUND**

The number of eigenvalues found to sufficient accuracy is zero.

**NE\_ZERO\_INIT\_RESID**

The option **Initial Residual** was selected but the starting vector held in **resid** is zero.

**NE\_ZERO\_SHIFT**

The option **Shifted Imaginary** or **Shifted Inverse Imaginary** has been selected and **sigmai** = zero on entry; **sigmai** must be non-zero for this mode of operation.

**7 Accuracy**

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

**8 Further Comments**

None.

**9 Example**

The example constructs the matrices  $A$  and  $B$  using LAPACK band storage format and solves  $Ax = \lambda Bx$  in shifted imaginary mode using the complex shift  $\sigma$ .

## 9.1 Program Text

```

/* nag_real_banded_sparse_eigensystem_sol (f12agc) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <naga02.h>
#include <nagf12.h>
#include <nagf16.h>

int main(void)
{
    /* Constants */
    Integer licomm=140;
    double rho = 100.0;

    /* Scalars */
    Complex res, eig;
    double alpha, h, resr, resi, sigmai, sigmar;
    Integer exit_status, i, j, k, kl, ksub, ksup, ku, lcomm;
    Integer ldab, n, nconv, ncv, nev, nx;
    /* Nag types */
    Nag_Boolean first;
    NagError fail;

    /* Arrays */
    double *ab=0, *ax=0, *comm=0, *eigvr=0, *eigvi=0, *eigest=0;
    double *mb=0, *mx=0, *resid=0, *v=0;
    Integer *icomm=0;

    exit_status = 0;
    INIT_FAIL(fail);

    Vprintf("nag_real_banded_sparse_eigensystem_sol (f12agc) Example Program"
           " Results\n");
    /* Skip heading in data file */
    Vscanf("%*[\n] ");

    Vscanf("%ld%ld%ld%lf%lf%*[\n] ", &nx, &nev, &ncv,
           &sigmar, &sigmai);
    n = nx * nx;
    /* ku, kl are number of superdiagonals and subdiagonals within the */
    /* band of matrices A and M. */
    kl = nx;
    ku = nx;
    /* ldab is the width of the band required to store the */
    /* factorisation of the matrix A. */
    ldab = 2*kl + ku + 1;
    lcomm = 60;
    /* Allocate memory. */
    if ( !(ab = NAG_ALLOC(ldab * n, double)) ||
         !(ax = NAG_ALLOC(n, double)) ||
         !(comm = NAG_ALLOC(lcomm, double)) ||
         !(eigvr = NAG_ALLOC(ncv, double)) ||
         !(eigvi = NAG_ALLOC(ncv, double)) ||
         !(eigest = NAG_ALLOC(ncv, double)) ||
         !(mb = NAG_ALLOC(ldab * n, double)) ||
         !(mx = NAG_ALLOC(n, double)) ||
         !(resid = NAG_ALLOC(n, double)) ||
         !(v = NAG_ALLOC(ncv * n, double)) ||
         !(icomm = NAG_ALLOC(licomm, Integer)) )
    {
        Vprintf("Allocation failure\n");
    }
}

```

```

        exit_status = -1;
        goto END;
    }
    /* Initialise communication arrays for problem using
       nag_real_banded_sparse_eigensystem_init (f12afc). */
    nag_real_banded_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 IMAGINARY", icomm, comm,
                                       &fail);

    /* Select the problem type using
       nag_real_sparse_eigensystem_option (f12adc). */
    nag_real_sparse_eigensystem_option("GENERALIZED", icomm, comm, &fail);

    /* Construct the matrix A in banded form and store in AB. */
    /* Zero out matrices first */
    for (j = 0; j < n*ldab; ++j)
    {
        ab[j] = 0.0;
        mb[j] = 0.0;
    }
    /* Main diagonal of A. */
    k = kl + ku;
    for (j = 0; j < n; ++j)
    {
        ab[k] = 4.;
        mb[k] = 4.;
        k = k + ldab;
    }
    /* First subdiagonal and superdiagonal of A. */
    ksup = kl + ku - 1;
    ksub = kl + ku + 1;
    h = .5 * rho / (double) (nx + 1);
    for (i = 1; i <= nx; ++i)
    {
        ksub = ksub + ldab;
        for (j = 1; j <= nx - 1; ++j)
        {
            ab[ksub] = - 1. + h;
            ab[ksup] = - 1. - h;
            ksup = ksup + ldab;
            ksub = ksub + ldab;
        }
        ksup = ksup + ldab;
    }

    ksub = kl + ku + 1 + ldab;
    ksup = kl + ku - 1;
    for (j = 1; j <= n - 1; ++j)
    {
        mb[ksup] = 1.;
        mb[ksub] = 1.;
        ksup = ksup + ldab;
        ksub = ksub + ldab;
    }
    /* KL-th subdiagonal and KU-th super-diagonal. */
    ksup = kl + nx*ldab;
    ksub = 2*kl + ku;
    for (i = 1; i <= nx - 1; ++i)
    {
        for (j = 1; j <= nx; ++j)
        {
            ab[ksup] = - 1.;
            ab[ksub] = - 1.;
            ksup = ksup + ldab;
            ksub = ksub + ldab;
        }
    }

    /* Find eigenvalues closest in value to SIGMA and corresponding

```

```

eigenvectors using nag_real_banded_sparse_eigensystem_sol
(f12agc). */
nag_real_banded_sparse_eigensystem_sol(kl, ku, ab, mb, sigmar,
                                       sigmai, &nconv, eigvr, eigvi,
                                       v, resid, v, comm, icomm,
                                       &fail);

if (fail.code == NE_NOERROR)
{
  /* Compute the residual norm ||A*x - lambda*Mx||. */
  first = Nag_TRUE;
  k = 0;
  for (j = 0; j <= nconv-1; ++j)
  {
    if (eigvi[j] == 0.)
    {
      /* Eigenvalue is Real. */
      /* ax <- AV_k, where V_k is kth Ritz vector. */
      /* Compute matrix-vector multiply using nag_dgbmv
         (f16pbc). */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &ab[kl], ldab, &v[k], 1, 0., ax, 1, &fail);
      /* mx <- MV_k. */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &mb[kl], ldab, &v[k], 1, 0., mx, 1, &fail);
      /* ax <- ax - (lambda_j) * mx. */
      alpha = -eigvr[j];
      /* Compute vector update using nag_daxpby (f16ecc). */
      nag_daxpby(n, alpha, mx, 1, 1., ax, 1, &fail);
      /* resr = norm(ax). */
      /* Compute 2-norm of Ritz estimates using nag_dge_norm
         (f16rac).*/
      nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, n, 1, ax,
                   n, &resr, &fail);
      /* Scale. */
      eigest[j] = resr / fabs(eigvr[j]);
    }
    else if (first)
    {
      /* First or a conjugate pair of eigenvalues. */

      /* Real part of residual Re[Ax-lambdaMx]. */
      /* ax <- AV_k, where V_k is real part of kth Ritz vector. */
      /* Compute matrix-vector multiply using nag_dgbmv
         (f16pbc). */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &ab[kl], ldab, &v[k], 1, 0., ax, 1, &fail);
      /* mx <- MV_k */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &mb[kl], ldab, &v[k], 1, 0., mx, 1, &fail);
      /* ax <- ax - (lambda_j).re * mx. */
      /* Compute vector update using nag_daxpby (f16ecc). */
      alpha = -eigvr[j];
      nag_daxpby(n, alpha, mx, 1, 1., ax, 1, &fail);
      /* mx <- MW_k where W_k is imaginary part kth Ritz vector. */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &mb[kl], ldab, &v[k+n], 1, 0., mx, 1, &fail);
      /* ax <- ax - (lambda_j).im * mx. */
      alpha = eigvi[j];
      nag_daxpby(n, alpha, mx, 1, 1., ax, 1, &fail);
      /* resr = norm(ax). */
      /* Compute 2-norm of Ritz estimates using nag_dge_norm
         (f16rac).*/
      nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, n, 1, ax,
                   n, &resr, &fail);
      /* Imaginary part of residual Im[Ax-lambdaMx]. */
      /* ax <- AW_k. */
      /* Compute matrix-vector multiply using nag_dgbmv
         (f16pbc). */
      nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
                &ab[kl], ldab, &v[k+n], 1, 0., ax, 1, &fail);
      /* ax <- ax - (lambda_j).re * mx. */

```

```

    alpha = -eigvr[j];
    /* Compute vector update using nag_daxpby (f16ecc). */
    nag_daxpby(n, alpha, mx, 1, 1., ax, 1, &fail);
    /* mx <- MV_k. */
    nag_dgbmv(Nag_ColMajor, Nag_NoTrans, n, n, kl, ku, 1.,
              &mb[kl], ldab, &v[k], 1, 0., mx, 1, &fail);
    /* ax <- ax - (lambda_j).im * mx. */
    alpha = -eigvi[j];
    nag_daxpby(n, alpha, mx, 1, 1., ax, 1, &fail);
    /* resi = norm(ax). */
    /* Compute 2-norm of Ritz estimates using nag_dge_norm
       (f16rac).*/
    nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, n, 1, ax,
                 n, &resi, &fail);
    /* Scale residual using Ritz value. */
    /* Assign to Complex type using nag_complex (a02bac) */
    res = nag_complex(resr, resi);
    eig = nag_complex(eigvr[j], eigvi[j]);
    eigest[j] = nag_complex_abs(res)/nag_complex_abs(eig);
    /* Set residual for second in conjugate pair. */
    eigest[j+1] = eigest[j];
    first = Nag_FALSE;
  }
  else
  {
    /* Second of complex conjugate pair; */
    /* Already got residual from first in pair. */
    first = Nag_TRUE;
  }
  k = k + n;
}

/* Print computed eigenvalues. */
Vprintf("\n The %4ld generalized Ritz values", nconv);
Vprintf(" closest to \n");
Vprintf(" ( %7.4f, +-%7.4f ) are:\n\n", sigmar, sigmai);
for (j = 0; j <= nconv-1; ++j)
{
  if (eigest[j] <= 1.0e-10)
  {
    Vprintf("%8ld%5s( %7.4f, %7.4f ).\n", j+1,
            "", eigvr[j], eigvi[j]);
  }
  else
  {
    Vprintf("%8ld%5s( %7.4f, %7.4f )%5s%18.16f.\n", j+1,
            "", eigvr[j], eigvi[j], " *** ", eigest[j]);
  }
}
}
else
{
  Vprintf(" Error from nag_real_banded_sparse_eigensystem_sol (f12agc)."\n\n"s\n", fail.message);
  exit_status = 1;
  goto END;
}
END:
if (ab) NAG_FREE(ab);
if (ax) NAG_FREE(ax);
if (comm) NAG_FREE(comm);
if (eigvr) NAG_FREE(eigvr);
if (eigvi) NAG_FREE(eigvi);
if (eigest) NAG_FREE(eigest);
if (mb) NAG_FREE(mb);
if (mx) NAG_FREE(mx);
if (resid) NAG_FREE(resid);
if (v) NAG_FREE(v);
if (icomm) NAG_FREE(icomm);

```

```
    return exit_status;  
}
```

## 9.2 Program Data

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) Example Program Data  
10 4 10 0.4 0.6 : Values for nx, nev, ncv, sigmar, sigmai

## 9.3 Program Results

nag\_real\_banded\_sparse\_eigensystem\_sol (f12agc) Example Program Results

The 4 generalized Ritz values closest to  
( 0.4000, +- 0.6000 ) are:

```
1      ( 0.3610, 0.7223 ).  
2      ( 0.3610, -0.7223 ).  
3      ( 0.4598, -0.7199 ).  
4      ( 0.4598, 0.7199 ).
```

---