NAG Library Function Document

nag_complex_banded_eigensystem_solve (f12auc)

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 the option setting function nag_complex_sparse_eigensystem_option (f12arc) need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 in nag_complex_sparse_eigensystem_option (f12arc) for a detailed description of the specification of the optional arguments.*

1 Purpose

nag_complex_banded_eigensystem_solve (f12auc) is the main solver function in a suite of functions consisting of nag_complex_sparse_eigensystem_option (f12arc), nag_complex_banded_eigensystem_init (f12atc) and nag_complex_banded_eigensystem_solve (f12auc). It must be called following an initial call to nag_complex_banded_eigensystem_init (f12atc) and following any calls to nag_complex_sparse_eigensystem_option (f12arc).

nag_complex_banded_eigensystem_solve (f12auc) returns approximations to selected eigenvalues, and (optionally) the corresponding eigenvectors, of a standard or generalized eigenvalue problem defined by complex banded non-Hermitian matrices. The banded matrix must be stored using the LAPACK column ordered storage format for complex banded non-Hermitian (see Section 3.3.4 in the f07 Chapter Introduction).

2 Specification

```
#include <nag.h>
#include <nagf12.h>
void nag_complex_banded_eigensystem_solve (Integer kl, Integer ku,
     const Complex ab[], const Complex mb[], Complex sigma, Integer *nconv,
     Complex d[], Complex z[], Complex resid[], Complex v[], Complex comm[],
     Integer icomm[], NagError *fail)
```

3 Description

The suite of functions is designed to calculate some of the eigenvalues, λ, (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, complex and non-Hermitian.

Following a call to the initialization function nag_complex_banded_eigensystem_init (f12atc), nag_complex_banded_eigensystem_solve (f12auc) returns the converged approximations to eigenvalues and (optionally) the corresponding approximate eigenvectors and/or a unitary basis for the associated approximate invariant subspace. The eigenvalues (and eigenvectors) are selected from those of a standard or generalized eigenvalue problem defined by complex banded non-Hermitian matrices. There is negligible additional computational cost to obtain eigenvectors; a unitary 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-Hermitian matrices; please refer to Section 3.3.4 in the f07 Chapter Introduction for details on this storage format.

nag_complex_banded_eigensystem_solve (f12auc) is based on the banded driver functions **znbdr1** to **znbdr4** 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-Hermitian matrices is provided in Lehoucq and Scott (1996). This suite of functions offers the same functionality as the ARPACK banded driver software for complex non-

Hermitian problems, but the interface design is quite different in order to make the option setting clearer and to combine the different drivers into a general purpose function.

nag_complex_banded_eigensystem_solve (f12auc), is a general purpose function that must be called following initialization by nag_complex_banded_eigensystem_init (f12atc). nag_complex_banded_eigensystem_solve (f12auc) uses options, set either by default or explicitly by calling nag_complex_sparse_eigensystem_option (f12arc), to return the converged approximations to selected eigenvalues and (optionally):

– the corresponding approximate eigenvectors;

– a unitary 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, Philidelphia

5 Arguments

Note: in the following description **n**, **nev** and **ncv** appears. In every case they should be interpretted as the value associated with the identically named argument in a prior call to nag_complex_banded_eigensystem_init (f12atc).

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 Complex *Input*

Note: the dimension, *dim*, of the array **ab** must be at least $\mathbf{n} \times (2 \times \mathbf{kl} + \mathbf{ku} + 1)$ (see nag_complex_banded_eigensystem_init (f12atc)).

On entry: must contain the matrix A in LAPACK column-ordered banded storage format for non-Hermitian matrices; that is, element a_{ij} is stored in $\mathbf{ab}[(j-1) \times (2 \times \mathbf{kl} + \mathbf{ku} + 1) + \mathbf{kl} + \mathbf{ku} + i - j]$, which may be written as $\mathbf{ab}[(2 \times j - 1) \times \mathbf{kl} + j \times \mathbf{ku} + i - 1]$, for $\max(1, j - \mathbf{ku}) \leq i \leq \min(n, j + \mathbf{kl})$ and $j = 1, 2, \ldots, n$, (see Section 3.3.4 in the f07 Chapter Introduction).

4: **mb**[dim] – const Complex *Input*

Note: the dimension, *dim*, of the array **mb** must be at least $\mathbf{n} \times (2 \times \mathbf{kl} + \mathbf{ku} + 1)$ (see nag_complex_banded_eigensystem_init (f12atc)).

On entry: must contain the matrix B in LAPACK column-ordered banded storage format for non-Hermitian matrices; that is, element a_{ij} is stored in $\mathbf{mb}[(j-1) \times (2 \times \mathbf{kl} + \mathbf{ku} + 1) + \mathbf{kl} + \mathbf{ku} + i - j]$, which may be written as

mb$[(2 \times j - 1) \times$ **kl** $+ j \times$ **ku** $+ i - 1]$, for $\max(1, j -$ **ku**$) \leq i \leq \min(n, j +$ **kl**$)$ and $j = 1, 2, \ldots, n$, (see Section 3.3.4 in the f07 Chapter Introduction).

5: **sigma** – Complex *Input*

On entry: if the **Shifted Inverse** mode (see nag_complex_sparse_eigensystem_option (f12arc)) has been selected then **sigma** must contain the shift used; otherwise **sigma** is not referenced. Section 4.2 in the f12 Chapter Introduction describes the use of shift and invert transformations.

6: **nconv** – Integer * *Output*

On exit: the number of converged eigenvalues.

7: **d**[**nev**] – Complex *Output*

On exit: the first **nconv** locations of the array **d** contain the converged approximate eigenvalues.

8: **z**[*dim*] – Complex *Output*

Note: the dimension, *dim*, of the array **z** must be at least **n** $\times$ **nev** if the default option **Vectors** = RITZ (see nag_complex_sparse_eigensystem_option (f12arc)) has been selected (see nag_complex_banded_eigensystem_init (f12atc)).

On exit: if the default option **Vectors** = RITZ (see nag_complex_sparse_eigensystem_option (f12arc)) has been selected then **z** contains the final set of eigenvectors corresponding to the eigenvalues held in **d**, otherwise **z** is not referenced and may be **NULL**. The complex eigenvector associated with an eigenvalue **d**$[j]$ is stored in the corresponding array section of **z**, namely **z**$[$**n** $\times (j - 1) + i - 1]$, for $i = 1, 2, \ldots,$ **n** and $j = 1, 2, \ldots,$ **nconv**.

9: **resid**[**n**] – Complex *Input/Output*

On entry: need not be set unless the option **Initial Residual** has been set in a prior call to nag_complex_sparse_eigensystem_option (f12arc) in which case **resid** must contain an initial residual vector.

On exit: contains the final residual vector. This can be used as the starting residual to improve convergence on the solution of a closely related eigenproblem. This has no relation to the error residual $Ax - \lambda x$ or $Ax - \lambda Bx$.

10: **v**[**n** $\times$ **ncv**] – Complex *Output*

On exit: if the option **Vectors** = SCHUR or RITZ (see nag_complex_sparse_eigensystem_option (f12arc)) has been set and a separate array **z** has been passed (i.e., **z** does not equal **v**), then the first **nconv** sections of **v**, of length n, will contain approximate Schur vectors that span the desired invariant subspace.

The jth Schur vector is stored in locations **v**$[$**n** $\times (j - 1) + i - 1]$, for $j = 1, 2, \ldots,$ **nconv** and $i = 1, 2, \ldots, n$.

11: **comm**[**60**] – Complex *Communication Array*

On entry: must remain unchanged from the prior call to nag_complex_sparse_eigensystem_option (f12arc) and nag_complex_banded_eigensystem_init (f12atc).

12: **icomm**[**140**] – Integer *Communication Array*

On entry: must remain unchanged from the prior call to nag_complex_sparse_eigensystem_option (f12arc) and nag_complex_banded_eigensystem_init (f12atc).

13: **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_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_EIGENVALUES

The number of eigenvalues found to sufficient accuracy is zero.

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

On entry, **kl** = ⟨*value*⟩.
Constraint: **kl** ≥ 0.

On entry, **ku** = ⟨*value*⟩.
Constraint: **ku** ≥ 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 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

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

The maximum number of iterations ≤ 0, the option **Iteration Limit** has been set to ⟨*value*⟩.

NE_NO_ARNOLDI_FAC

Could not build an Arnoldi factorization. The size of the current Arnoldi factorization = ⟨*value*⟩.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_NO_SHIFTS_APPLIED

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

NE_OPT_INCOMPAT

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

NE_OVERFLOW

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

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 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_TOO_MANY_ITER

The maximum number of iterations has been reached. The maximum number of iterations = ⟨*value*⟩. The number of converged eigenvalues = ⟨*value*⟩.

NE_ZERO_RESID

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

7 Accuracy

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

8 Parallelism and Performance

nag_complex_banded_eigensystem_solve (f12auc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_complex_banded_eigensystem_solve (f12auc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

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

10.1 Program Text

```
/* nag_complex_banded_eigensystem_solve (f12auc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf12.h>
#include <nagf16.h>
#include <nagx04.h>

#define AB(I, J) ab[(J-1)*pdab + kl + ku + I - J]
#define MB(I,J) mb[(J-1)*pdab + kl + ku + I - J]

int main(void)
{

  Integer exit_status = 0;
  Complex one = {1.0, 0.0} ;
  Complex minusone = {-1.0, 0.0} ;
  Complex zero = {0.0, 0.0} ;
  Nag_Boolean print_res = Nag_FALSE;
  /* Scalars */
  Complex ch_sub, ch_sup, alpha, sigma;
  double h, rho, anorm;
  Integer j, kl, ku, lcomm, licomm, n, ncol, nconv, ncv, nev, nx;
  Integer pdab, pdmb, pdv;
  /* Arrays */
  Complex commd[1];
  Integer icommd[1];
  Complex  *ab = 0, *ax = 0, *comm = 0, *d = 0, *mb = 0;
  Complex  *mx = 0, *resid = 0, *v = 0;
  double  *d_print = 0;
  Integer  *icomm = 0;
  /* NAG types */
  NagError    fail;
  Nag_OrderType             order = Nag_ColMajor;
  Nag_MatrixType            matrix = Nag_GeneralMatrix;
  Nag_DiagType              diag = Nag_NonUnitDiag;
  Nag_TransType             trans = Nag_NoTrans;
  INIT_FAIL(fail);

  printf("nag_complex_banded_eigensystem_solve (f12auc) Example Program "
         "Results\n\n");
  fflush(stdout);
  /* Skip heading in data file*/
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read mesh size, number of eigenvalues wanted and size of subspace. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &nx, &nev, &ncv);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &nx, &nev, &ncv);
#endif
  /* Read complex value close to which eigenvalues are sought. */
#ifdef _WIN32
```

```
  scanf_s(" ( %lf , %lf ) %*[^\n] ", &sigma.re, &sigma.im);
#else
  scanf(" ( %lf , %lf ) %*[^\n] ", &sigma.re, &sigma.im);
#endif

  n = nx * nx;
  /* Initialize for the solution of a complex banded eigenproblem using
   * nag_complex_banded_eigensystem_init (f12atc).
   * But first get the required array lengths using arrays of length 1
   * to store required lengths.
   */
  licomm = -1;
  lcomm = -1;
  nag_complex_banded_eigensystem_init(n, nev, ncv, icommd, licomm, commd, lcomm,
                                      &fail);
  licomm = icommd[0];
  lcomm = (Integer)(commd[0].re);
  if (
      !(icomm = NAG_ALLOC((MAX(1, licomm)), Integer))||
      !(comm = NAG_ALLOC((MAX(1, lcomm)), Complex))
       )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  nag_complex_banded_eigensystem_init(n, nev, ncv, icomm, licomm, comm, lcomm,
                                      &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_sparse_eigensystem_init (f12anc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* Set options to show that the problem is a generalized eigenproblem and
   * that eigenvalues are to be computed using shifted inverses using
   * nag_complex_sparse_eigensystem_option (f12arc).
   */
  nag_complex_sparse_eigensystem_option("SHIFTED INVERSE", icomm, comm, &fail);
  nag_complex_sparse_eigensystem_option("GENERALIZED", icomm, comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_sparse_eigensystem_option (f12arc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }

  /* Construct the matrix A in banded form and store in AB.
   * KU, KL are number of superdiagonals and subdiagonals within
   * the band of matrices A and M.
   */
  kl = nx;
  ku = nx;
  pdab = 2 * kl + ku + 1;
  pdmb = pdab;
  if (!(ab = NAG_ALLOC(pdab*n, Complex))||
      !(mb = NAG_ALLOC(pdmb*n, Complex))
      )
    {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }

  /* Zero out AB and MB.*/
  for (j=0; j<pdab*n; j++)
    {
      ab[j] = zero;
```

```
      mb[j] = zero;
    }

  rho = 100.0;
  h = 1.0/(double)(nx + 1);
  /* assign complex values using nag_complex (a02bac). */
  ch_sub = nag_complex(0.5 * h * rho - 1.0, 0.0);
  ch_sup = nag_complex(-0.5 * h * rho - 1.0, 0.0);

  /* Set main diagonal of A and B to (4,0);
   * first subdiagonal and superdiagonal of A are (-1 +/- rho/2h,0)..
   * Outermost subdiagonal and super-diagonal are (-1,0).
   */
  for ( j=1; j<=n; j++)
    {
      AB(j,j) = nag_complex(4.0, 0.0);
      MB(j,j) = nag_complex(4.0, 0.0);
      if ((j-1)%nx!=0 && j!=n)
        {
          AB(j+1,j) = ch_sub;
        }
      if (j%nx!=0 && j!=1)
        {
          AB(j-1,j) = ch_sup;
        }
      if (j<=n-kl)
        AB(j+kl,j) = minusone;
      if (j>ku)
        AB(j-ku,j) = minusone;
    }
  for (j=2; j<n; j++)
    {
      MB(j+1,j) =  one;
    }
  for (j=2; j<n; j++)
    {
      MB(j-1,j) =  one;
    }

  /* Allocate v to hold eigenvectors and d to hold eigenvalues. */
  pdv = n;
  if (
      !(d = NAG_ALLOC((nev), Complex))||
      !(v = NAG_ALLOC((pdv)*(ncv), Complex))||
      !(resid = NAG_ALLOC((n), Complex))
       )
    {
      printf("Allocation failure\n");
      exit_status = -3;
      goto END;
    }
  /* Find eigenvalues closest in value to sigma and corresponding
   * eigenvectors using nag_complex_banded_eigensystem_solve (f12auc).
   */
  nag_complex_banded_eigensystem_solve(kl, ku, ab, mb, sigma, &nconv, d,
                                       v, resid, v, comm, icomm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_complex_banded_eigensystem_solve (f12auc).\n%s\n",
              fail.message);
      exit_status = 3;
      goto END;
    }

  /* d_print stores eigenvalues (and possibly residuals) as real array. */
  if (!(d_print = NAG_ALLOC(nconv*3, double)))
    {
      printf("Allocation failure\n");
      exit_status = -4;
      goto END;
    }
```

```
  for (j=0;j<nconv; j++)
    {
      d_print[j] =  (d[j]).re;
      d_print[j+nconv] =  (d[j]).im;
    }
  /* By default this example will not calculate and print residuals.
   * It will, however if the following line is uncommented.
   */
  /* print_res = Nag_TRUE; */
  if (print_res) {
    ncol = 3;
    /* Compute the residual norm  ||A*x - lambda*B*x||.
     * Matrix vector products Ax and Bx, for complex banded A and B, are formed
     * using nag_zgbmv (f16sbc).
     * The scaled vector subtraction is performed using nag_zaxpby (f16gcc).
     */
    if (
        !(ax = NAG_ALLOC(n, Complex))||
        !(mx = NAG_ALLOC(n, Complex))
         )
      {
        printf("Allocation failure\n");
        exit_status = -4;
        goto END;
      }
    for ( j=0; j<nconv; j++) {
      nag_zgbmv(order, trans, n, n, kl, ku, one, &ab[kl], pdab, &v[pdv*j],
                1, zero, ax, 1, &fail);
      if (fail.code == NE_NOERROR) {
        nag_zgbmv(order, trans, n, n, kl, ku, one, &mb[kl], pdmb, &v[pdv*j],
                  1, zero, mx, 1, &fail);
        if (fail.code == NE_NOERROR) {
          /* alpha holds -d[j] using nag_complex_negate (a02cec). */
          alpha = nag_complex_negate(d[j]);
          nag_zaxpby(n, alpha, mx, 1, one, ax, 1, &fail);
        }
      }
      if (fail.code != NE_NOERROR) {
        printf("Error in calculating residual norm.\n%s\n", fail.message);
        exit_status = 4;
        goto END;
      }
      /* Get ||ax|| = ||Ax - lambda Bx|| using nag_zge_norm (f16uac)
       * with the Frobenius norm on a complex general matrix with one column.
       */
      nag_zge_norm(order, Nag_FrobeniusNorm, n, 1, ax, n, &anorm, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
          exit_status = 5;
          goto END;
        }
      /* nag_complex_abs (a02dbc) returns the modulus of the eigenvalue */
      d_print[j+2*nconv] = anorm/nag_complex_abs(d[j]);
    }
  } else {
    ncol = 2;
  }
  printf("\n");

  /* Print the eigenvalues (and possibly the residuals) using the matrix
   * printer, nag_gen_real_mat_print (x04cac).
   */
  printf("  Number of eigenvalues wanted     = %"NAG_IFMT"\n",nev);
  printf("  Number of eigenvalues converged  = %"NAG_IFMT"\n",nconv);
  printf("  Eigenvalues are closest to Sigma = ( %lf , %lf )\n\n",
         sigma.re, sigma.im);
  fflush(stdout);
  nag_gen_real_mat_print(order, matrix, diag, nconv, ncol, d_print, nconv,
                         " Ritz values closest to sigma", 0, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
      exit_status = 6;
    }
 END:
  NAG_FREE(ab);
  NAG_FREE(ax);
  NAG_FREE(comm);
  NAG_FREE(d);
  NAG_FREE(mb);
  NAG_FREE(mx);
  NAG_FREE(resid);
  NAG_FREE(v);
  NAG_FREE(d_print);
  NAG_FREE(icomm);
  return exit_status;
}
```

10.2 Program Data

```
nag_complex_banded_eigensystem_solve (f12auc) Example Program Data
  10    4    10              : nx nev ncv
 ( 0.4, 0.6)                 : sigma
```

10.3 Program Results

```
nag_complex_banded_eigensystem_solve (f12auc) Example Program Results


  Number of eigenvalues wanted     = 4
  Number of eigenvalues converged  = 4
  Eigenvalues are closest to Sigma = ( 0.400000 , 0.600000 )

  Ritz values closest to sigma
         1       2
 1   0.3610  0.7223
 2   0.4598  0.7199
 3   0.2868  0.7241
 4   0.2410  0.7257
```