NAG Library Function Document

nag_zheevr (f08frc)

1 Purpose

nag_zheevr (f08frc) computes selected eigenvalues and, optionally, eigenvectors of a complex n by n Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zheevr (Nag_OrderType order, Nag_JobType job, Nag_RangeType range,
     Nag_UploType uplo, Integer n, Complex a[], Integer pda, double vl,
     double vu, Integer il, Integer iu, double abstol, Integer *m,
     double w[], Complex z[], Integer pdz, Integer isuppz[], NagError *fail)
```

3 Description

The Hermitian matrix is first reduced to a real tridiagonal matrix T, using unitary similarity transformations. Then whenever possible, nag_zheevr (f08frc) computes the eigenspectrum using Relatively Robust Representations. nag_zheevr (f08frc) computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various 'good' LDL^{T} representations (also known as Relatively Robust Representations). Gram–Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the ith unreduced block of T:

(a) compute $T - \sigma_i I = L_i D_i L_i^{\mathrm{T}}$, such that $L_i D_i L_i^{\mathrm{T}}$ is a relatively robust representation,

(b) compute the eigenvalues, λ_j, of $L_i D_i L_i^{\mathrm{T}}$ to high relative accuracy by the dqds algorithm,

(c) if there is a cluster of close eigenvalues, 'choose' σ_i close to the cluster, and go to (a),

(d) given the approximate eigenvalue λ_j of $L_i D_i L_i^{\mathrm{T}}$, compute the corresponding eigenvector by forming a rank-revealing twisted factorization.

The desired accuracy of the output can be specified by the argument **abstol**. For more details, see Dhillon (1997) and Parlett and Dhillon (2000).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Barlow J and Demmel J W (1990) Computing accurate eigensystems of scaled diagonally dominant matrices *SIAM J. Numer. Anal.* **27** 762–791

Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

Dhillon I (1997) A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem *Computer Science Division Technical Report No. UCB//CSD-97-971* UC Berkeley

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Parlett B N and Dhillon I S (2000) Relatively robust representations of symmetric tridiagonals *Linear Algebra Appl.* **309** 121–151

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **job** – Nag_JobType *Input*

On entry: indicates whether eigenvectors are computed.

job = Nag_EigVals
Only eigenvalues are computed.

job = Nag_DoBoth
Eigenvalues and eigenvectors are computed.

Constraint: **job** = Nag_EigVals or Nag_DoBoth.

3: **range** – Nag_RangeType *Input*

On entry: if **range** = Nag_AllValues, all eigenvalues will be found.

If **range** = Nag_Interval, all eigenvalues in the half-open interval $(\mathbf{vl}, \mathbf{vu}]$ will be found.

If **range** = Nag_Indices, the **il**th to **iu**th eigenvalues will be found.

For **range** = Nag_Interval or Nag_Indices and $\mathbf{iu} - \mathbf{il} < \mathbf{n} - 1$, nag_dstebz (f08jjc) and nag_zstein (f08jxc) are called.

Constraint: **range** = Nag_AllValues, Nag_Interval or Nag_Indices.

4: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangular part of A is stored.

If **uplo** = Nag_Lower, the lower triangular part of A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

5: **n** – Integer *Input*

On entry: n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

6: **a**$[dim]$ – Complex *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.

On entry: the n by n Hermitian matrix A.

If **order** = Nag_ColMajor, A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = Nag_RowMajor, A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag_Lower, the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: the lower triangle (if **uplo** = Nag_Lower) or the upper triangle (if **uplo** = Nag_Upper) of **a**, including the diagonal, is overwritten.

7: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraint: **pda** $\geq \max(1, \mathbf{n})$.

8: **vl** – double *Input*

9: **vu** – double *Input*

On entry: if **range** = Nag_Interval, the lower and upper bounds of the interval to be searched for eigenvalues.

If **range** = Nag_AllValues or Nag_Indices, **vl** and **vu** are not referenced.

Constraint: if **range** = Nag_Interval, **vl** $<$ **vu**.

10: **il** – Integer *Input*

11: **iu** – Integer *Input*

On entry: if **range** = Nag_Indices, the indices (in ascending order) of the smallest and largest eigenvalues to be returned.

If **range** = Nag_AllValues or Nag_Interval, **il** and **iu** are not referenced.

Constraints:

if **range** = Nag_Indices and $\mathbf{n} = 0$, $\mathbf{il} = 1$ and $\mathbf{iu} = 0$;
if **range** = Nag_Indices and $\mathbf{n} > 0$, $1 \leq \mathbf{il} \leq \mathbf{iu} \leq \mathbf{n}$.

12: **abstol** – double *Input*

On entry: the absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval $[a, b]$ of width less than or equal to

$$\mathbf{abstol} + \epsilon \max(|a|, |b|),$$

where ϵ is the ***machine precision***. If **abstol** is less than or equal to zero, then $\epsilon \|T\|_1$ will be used in its place, where T is the real tridiagonal matrix obtained by reducing A to tridiagonal form. See Demmel and Kahan (1990).

If high relative accuracy is important, set **abstol** to nag_real_safe_small_number( ), although doing so does not currently guarantee that eigenvalues are computed to high relative accuracy. See Barlow and Demmel (1990) for a discussion of which matrices can define their eigenvalues to high relative accuracy.

13: **m** – Integer * *Output*

On exit: the total number of eigenvalues found. $0 \leq \mathbf{m} \leq \mathbf{n}$.

If **range** = Nag_AllValues, $\mathbf{m} = \mathbf{n}$.

If **range** = Nag_Indices, $\mathbf{m} = \mathbf{iu} - \mathbf{il} + 1$.

14: **w**[dim] – double *Output*

Note: the dimension, *dim*, of the array **w** must be at least $\max(1, \mathbf{n})$.

On exit: the first **m** elements contain the selected eigenvalues in ascending order.

15: **z**[dim] – Complex *Output*

Note: the dimension, *dim*, of the array **z** must be at least

$\max(1, \mathbf{pdz} \times \mathbf{n})$ when **job** = Nag_DoBoth;
1 otherwise.

The (i,j)th element of the matrix Z is stored in

$\mathbf{z}[(j-1) \times \mathbf{pdz} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{z}[(i-1) \times \mathbf{pdz} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **job** = Nag_DoBoth, the first **m** columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with $\mathbf{w}[i-1]$.

If **job** = Nag_EigVals, **z** is not referenced.

16: **pdz** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **z**.

Constraints:

if **job** = Nag_DoBoth, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

17: **isuppz**[*dim*] – Integer *Output*

Note: the dimension, *dim*, of the array **isuppz** must be at least $\max(1, 2 \times \mathbf{m})$.

On exit: the support of the eigenvectors in **z**, i.e., the indices indicating the nonzero elements in **z**. The ith eigenvector is nonzero only in elements $\mathbf{isuppz}[2 \times i - 2]$ through $\mathbf{isuppz}[2 \times i - 1]$. Implemented only for **range** = Nag_AllValues or Nag_Indices and $\mathbf{iu} - \mathbf{il} = \mathbf{n} - 1$.

18: **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_CONVERGENCE

nag_zheevr (f08frc) failed to converge.

NE_ENUM_INT_2

On entry, **job** = ⟨*value*⟩, **pdz** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **job** = Nag_DoBoth, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

NE_ENUM_INT_3

On entry, **range** = ⟨*value*⟩, **il** = ⟨*value*⟩, **iu** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **range** = Nag_Indices and **n** = 0, **il** = 1 and **iu** = 0;
if **range** = Nag_Indices and **n** > 0, $1 \leq \mathbf{il} \leq \mathbf{iu} \leq \mathbf{n}$.

NE_ENUM_REAL_2

On entry, **range** = ⟨*value*⟩, **vl** = ⟨*value*⟩ and **vu** = ⟨*value*⟩.
Constraint: if **range** = Nag_Interval, **vl** < **vu**.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

On entry, $\mathbf{pdz} = \langle value \rangle$.
Constraint: $\mathbf{pdz} > 0$.

NE_INT_2

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

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_NO_LICENCE

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

7 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrix $(A + E)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the ***machine precision***. See Section 4.7 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_zheevr (f08frc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zheevr (f08frc) 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

The total number of floating-point operations is proportional to n^3.

The real analogue of this function is nag_dsyevr (f08fdc).

10 Example

This example finds the eigenvalues with indices in the range $[2,3]$, and the corresponding eigenvectors, of the Hermitian matrix

$$A = \begin{pmatrix} 1 & 2-i & 3-i & 4-i \\ 2+i & 2 & 3-2i & 4-2i \\ 3+i & 3+2i & 3 & 4-3i \\ 4+i & 4+2i & 4+3i & 4 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_zheevr (f08frc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  double        abstol, vl = 0.0, vu = 0.0;
  Integer       i, il, iu, j, m, n, pda, pdz;
  Integer       exit_status = 0;
  /* Arrays */
  Complex       *a = 0, *z = 0;
  double        *w = 0;
  Integer       *isuppz = 0;
  /* Nag Types */
  Nag_OrderType order;
  NagError      fail;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J - 1) * pda + I - 1]
#define Z(I, J) z[(J - 1) * pdz + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
#define Z(I, J) z[(I - 1) * pdz + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zheevr (f08frc) Example Program Results\n\n");

  /* Skip heading in data file and read n and the lower and upper
   * indices of the smallest and largest eigenvalues to be found.
   */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &il, &iu);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &il, &iu);
#endif

  m = iu - il + 1;
```

```
  pda = n;
  pdz = n;

  /* Allocate memory */
  if (!(a = NAG_ALLOC(n*n, Complex)) ||
      !(z = NAG_ALLOC(n*pdz, Complex)) ||
      !(w = NAG_ALLOC(n, double)) ||
      !(isuppz = NAG_ALLOC(2*pdz, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the upper triangular part of the matrix A from data file */
  for (i = 1; i <= n; ++i)
    for (j = i; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
      scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Set the absolute error tolerance for eigenvalues.
   * With abstol set to zero, the default value is used instead.
   */
  abstol = 0.0;

  /* nag_zheevr (f08frc).
   * Solve the symmetric eigenvalue problem.
   */
  nag_zheevr(order, Nag_DoBoth, Nag_Indices, Nag_Upper, n, a, pda, vl, vu, il,
             iu, abstol, &m, w, z, pdz, isuppz, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zheevr (f08frc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_complex_divide (a02cdc).
   * Normalize the eigenvectors.
   */
  for(j=1; j<=m; j++)
    for(i=n; i>=1; i--)
      Z(i, j) = nag_complex_divide(Z(i, j),Z(1, j));

  /* Print solution */
  printf("Selected eigenvalues\n");
  for (j = 0; j < m; ++j)
    printf("%8.4f%s", w[j], (j+1)%8 == 0?"\n":" ");
  printf("\n");

  /* nag_gen_complx_mat_print (x04dac).
   * Print selected eigenvectors.
   */
  fflush(stdout);
  nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                           m, z, pdz, "Selected eigenvectors", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
```

```
 END:
  NAG_FREE(a);
  NAG_FREE(z);
  NAG_FREE(w);
  NAG_FREE(isuppz);

  return exit_status;
}

#undef A
#undef Z
```

10.2 Program Data

```
nag_zheevr (f08frc) Example Program Data

   4           2           3                          :Values of n, il and iu

  (1.0, 0.0)  (2.0,-1.0)  (3.0,-1.0)  (4.0,-1.0)
              (2.0, 0.0)  (3.0,-2.0)  (4.0,-2.0)
                          (3.0, 0.0)  (4.0,-3.0)
                                      (4.0, 0.0)     :End of matrix A
```

10.3 Program Results

```
nag_zheevr (f08frc) Example Program Results

Selected eigenvalues
 -0.6886   1.1412
 Selected eigenvectors
              1           2
 1      1.0000      1.0000
        0.0000     -0.0000

 2     -0.7703      0.0516
       -0.1746      1.2795

 3      0.4559     -1.1962
        0.4892     -0.2954

 4     -0.3464      0.7876
       -0.4448     -0.5075
```