NAG C Library Function Document

nag_zpteqr (f08juc)

1 Purpose

nag_zpteqr (f08juc) computes all the eigenvalues, and optionally all the eigenvectors, of a complex Hermitian positive-definite matrix which has been reduced to tridiagonal form.

2 Specification

```
void nag_zpteqr (Nag_OrderType order, Nag_ComputeZType compz, Integer n,
     double d[], double e[], Complex z[], Integer pdz, NagError *fail)
```

3 Description

nag_zpteqr (f08juc) computes all the eigenvalues, and optionally all the eigenvectors, of a real symmetric positive-definite tridiagonal matrix T. In other words, it can compute the spectral factorization of T as

$$T = Z\Lambda Z^T,$$

where Λ is a diagonal matrix whose diagonal elements are the eigenvalues λ_i, and Z is the orthogonal matrix whose columns are the eigenvectors z_i. Thus

$$Tz_i = \lambda_i z_i, \quad i = 1, 2, \ldots, n.$$

The function stores the real orthogonal matrix Z in a ***complex*** array, so that it may be used to compute all the eigenvalues and eigenvectors of a complex Hermitian positive-definite matrix A which has been reduced to tridiagonal form T:

$$\begin{aligned} A &= QTQ^H, \quad \text{where } Q \text{ is unitary} \\ &= (QZ)\Lambda(QZ)^H. \end{aligned}$$

In this case, the matrix Q must be formed explicitly and passed to nag_zpteqr (f08juc), which is called with **compz** = **Nag_UpdateZ**. The functions which must be called to perform the reduction to tridiagonal form and form Q are:

full matrix	nag_zhetrd (f08fsc) + nag_zungtr (f08ftc)
full matrix, packed storage	nag_zhptrd (f08gsc) + nag_zupgtr (f08gtc)
band matrix	nag_zhbtrd (f08hsc) with **vect** = **Nag_FormQ**.

nag_zpteqr (f08juc) first factorizes T as LDL^H where L is unit lower bidiagonal and D is diagonal. It forms the bidiagonal matrix $B = LD^{\frac{1}{2}}$, and then calls nag_zbdsqr (f08msc) to compute the singular values of B which are the same as the eigenvalues of T. The method used by the function allows high relative accuracy to be achieved in the small eigenvalues of T. The eigenvectors are normalized so that $\|z_i\|_2 = 1$, but are determined only to within a complex factor of absolute value 1.

4 References

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

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

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

2: **compz** – Nag_ComputeZType *Input*

On entry: indicates whether the eigenvectors are to be computed as follows:

if **compz** = **Nag_NotZ**, only the eigenvalues are computed (and the array **z** is not referenced);

if **compz** = **Nag_InitZ**, the eigenvalues and eigenvectors of T are computed (and the array **z** is initialised by the routine);

if **compz** = **Nag_UpdateZ**, the eigenvalues and eigenvectors of A are computed (and the array **z** must contain the matrix Q on entry).

Constraint: **compz** = **Nag_NotZ**, **Nag_UpdateZ** or **Nag_InitZ**.

3: **n** – Integer *Input*

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

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

4: **d**[dim] – double *Input/Output*

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

On entry: the diagonal elements of the tridiagonal matrix T.

On exit: the n eigenvalues in descending order, unless **fail** > 0, in which case the array is overwritten.

5: **e**[dim] – double *Input/Output*

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

On entry: the off-diagonal elements of the tridiagonal matrix T.

On exit: the array is overwritten.

6: **z**[dim] – Complex *Input/Output*

Note: the dimension, dim, of the array **z** must be at least
$\max(1, \mathbf{pdz} \times \mathbf{n})$ when **compz** = **Nag_UpdateZ** or **Nag_InitZ**;
1 when **compz** = **Nag_NotZ**.

If **order** = **Nag_ColMajor**, the (i,j)th element of the matrix Z is stored in $\mathbf{z}[(j-1) \times \mathbf{pdz} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i,j)th element of the matrix Z is stored in $\mathbf{z}[(i-1) \times \mathbf{pdz} + j - 1]$.

On entry: if **compz** = **Nag_UpdateZ**, **z** must contain the unitary matrix Q from the reduction to tridiagonal form. If **compz** = **Nag_InitZ**, **z** need not be set.

On exit: if **compz** = **Nag_InitZ** or **Nag_UpdateZ**, the n required orthonormal eigenvectors stored as columns of z; the ith column corresponds to the ith eigenvalue, where $i = 1, 2, \ldots, n$, unless **fail** > 0.

z is not referenced if **compz** = **Nag_NotZ**.

7: **pdz** – Integer *Input*

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

Constraints:

if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, $\mathbf{pdz} \geq \max(1, \mathbf{n})$;

if **compz** = **Nag_NotZ**, **pdz** ≥ 1.

8: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

On entry, **pdz** = ⟨*value*⟩.
Constraint: **pdz** > 0.

NE_ENUM_INT_2

On entry, **compz** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdz** = ⟨*value*⟩.
Constraint: if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, **pdz** $\geq \max(1, \mathbf{n})$;
if **compz** = **Nag_NotZ**, **pdz** ≥ 1.

NE_CONVERGENCE

The leading minor of order ⟨*value*⟩ is not positive-definite and the Cholesky factorization of T could not be completed. Hence T itself is not positive-definite.

The algorithm to compute the singular values of the Cholesky factor B failed to converge; ⟨*value*⟩ off-diagonal elements did not converge to zero.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ 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.

7 Accuracy

The eigenvalues and eigenvectors of T are computed to high relative accuracy which means that if they vary widely in magnitude, then any small eigenvalues (and corresponding eigenvectors) will be computed more accurately than, for example, with the standard QR method. However, the reduction to tridiagonal form (prior to calling the function) may exclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix if its eigenvalues vary widely in magnitude.

To be more precise, let H be the tridiagonal matrix defined by $H = DTD$, where D is diagonal with $d_{ii} = t_{ii}^{-\frac{1}{2}}$, and $h_{ii} = 1$ for all i. If λ_i is an exact eigenvalue of T and $\tilde{\lambda}_i$ is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\kappa_2(H)\lambda_i$$

where $c(n)$ is a modestly increasing function of n, ϵ is the ***machine precision***, and $\kappa_2(H)$ is the condition number of H with respect to inversion defined by: $\kappa_2(H) = \|H\| \cdot \|H^{-1}\|$.

If z_i is the corresponding exact eigenvector of T, and $\tilde{z}_i$ is the corresponding computed eigenvector, then the angle $\theta(\tilde{z}_i, z_i)$ between them is bounded as follows:

$$\theta(\tilde{z}_i, z_i) \leq \frac{c(n)\epsilon\kappa_2(H)}{relgap_i}$$

where $relgap_i$ is the relative gap between λ_i and the other eigenvalues, defined by

$$relgap_i = \min_{i \neq j} \frac{|\lambda_i - \lambda_j|}{(\lambda_i + \lambda_j)}.$$

8 Further Comments

The total number of real floating-point operations is typically about $30n^2$ if **compz** = **Nag_NotZ** and about $12n^3$ if **compz** = **Nag_UpdateZ** or **Nag_InitZ**, but depends on how rapidly the algorithm converges. When **compz** = **Nag_NotZ**, the operations are all performed in scalar mode; the additional operations to compute the eigenvectors when **compz** = **Nag_UpdateZ** or **Nag_InitZ** can be vectorized and on some machines may be performed much faster.

The real analogue of this function is nag_dpteqr (f08jgc).

9 Example

To compute all the eigenvalues and eigenvectors of the complex Hermitian positive-definite matrix A, where

$$A = \begin{pmatrix} 6.02 + 0.00i & -0.45 + 0.25i & -1.30 + 1.74i & 1.45 - 0.66i \\ -0.45 - 0.25i & 2.91 + 0.00i & 0.05 + 1.56i & -1.04 + 1.27i \\ -1.30 - 1.74i & 0.05 - 1.56i & 3.29 + 0.00i & 0.14 + 1.70i \\ 1.45 + 0.66i & -1.04 - 1.27i & 0.14 - 1.70i & 4.18 + 0.00i \end{pmatrix}.$$

9.1 Program Text

```
/* nag_zpteqr (f08juc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer  i, j, n, pda, pdz, d_len, e_len, tau_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_UploType  uplo;
  Nag_OrderType order;
  /* Arrays */
  char    uplo_char[2];
  Complex *a=0, *tau=0, *z=0;
  double  *d=0, *e=0;

#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);
  Vprintf("f08juc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);

  pda = n;
  pdz = n;
  tau_len = n-1;
  d_len = n;
  e_len = n-1;
  /* Allocate memory */
  if ( !(a = NAG_ALLOC(n * n, Complex)) ||
       !(tau = NAG_ALLOC(tau_len, Complex)) ||
       !(z = NAG_ALLOC(n * n, Complex)) ||
       !(d = NAG_ALLOC(d_len, double)) ||
       !(e = NAG_ALLOC(e_len, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A from data file */
  Vscanf(" ' %1s '%*[^\n] ", uplo_char);
  if (*(unsigned char *)uplo_char == 'L')
    uplo = Nag_Lower;
  else if (*(unsigned char *)uplo_char == 'U')
    uplo = Nag_Upper;
  else
    {
      Vprintf("Unrecognised character for Nag_UploType type\n");
      exit_status = -1;
      goto END;
    }
  if (uplo == Nag_Upper)
    {

      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
        }
      Vscanf("%*[^\n] ");
    }

  /* Reduce A to tridiagonal form T = (Q**H)*A*Q */
  f08fsc(order, uplo, n, a, pda, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08fsc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Copy A into Z */
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
```

```
        {
          for (j = i; j <= n; ++j)
            {
              Z(i,j).re = A(i,j).re;
              Z(i,j).im = A(i,j).im;
            }
        }
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            {
              Z(i,j).re = A(i,j).re;
              Z(i,j).im = A(i,j).im;
            }
        }
    }

  /* Form Q explicitly, storing the result in Z */
  f08ftc(order, uplo, n, z, pdz, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08ftc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Calculate all the eigenvalues and eigenvectors of A */
  f08juc(order, Nag_UpdateZ, n, d, e, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08juc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print eigenvalues and eigenvectors */
  Vprintf(" Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    Vprintf("%7.4f%s", d[i-1], i%4==0 ?"\n":"            ");
  Vprintf("\n");
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         z, pdz, Nag_BracketForm, "%7.4f", "Eigenvectors",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0,
         0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (a) NAG_FREE(a);
  if (tau) NAG_FREE(tau);
  if (z) NAG_FREE(z);
  if (d) NAG_FREE(d);
  if (e) NAG_FREE(e);

  return exit_status;
}
```

9.2 Program Data

```
f08juc Example Program Data
  4                                                         :Value of N
  'L'                                                       :Value of UPLO
 ( 6.02, 0.00)
 (-0.45,-0.25) ( 2.91, 0.00)
 (-1.30,-1.74) ( 0.05,-1.56) ( 3.29, 0.00)
 ( 1.45, 0.66) (-1.04,-1.27) ( 0.14,-1.70) ( 4.18, 0.00)   :End of matrix A
```

9.3 Program Results

```
f08juc Example Program Results

 Eigenvalues
 7.9995             5.9976             2.0003             0.4026

 Eigenvectors
                   1                  2                  3                  4
 1  ( 0.7289, 0.0000)  ( 0.2001, 0.4724)  (-0.2133, 0.1498)  ( 0.0995,-0.3573)
 2  (-0.1651,-0.2067)  (-0.2461, 0.3742)  ( 0.7308, 0.0000)  ( 0.2867,-0.3364)
 3  (-0.4170,-0.1413)  ( 0.4476, 0.1455)  (-0.3282, 0.0471)  ( 0.6890, 0.0000)
 4  ( 0.1748, 0.4175)  ( 0.5610, 0.0000)  ( 0.5203, 0.1317)  ( 0.0659, 0.4336)
```