NAG Library Function Document

nag_zhgeqz (f08xsc)

1 Purpose

nag_zhgeqz (f08xsc) implements the QZ method for finding generalized eigenvalues of the complex matrix pair (A, B) of order n, which is in the generalized upper Hessenberg form.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zhgeqz (Nag_OrderType order, Nag_JobType job,
     Nag_ComputeQType compq, Nag_ComputeZType compz, Integer n, Integer ilo,
     Integer ihi, Complex a[], Integer pda, Complex b[], Integer pdb,
     Complex alpha[], Complex beta[], Complex q[], Integer pdq, Complex z[],
     Integer pdz, NagError *fail)
```

3 Description

nag_zhgeqz (f08xsc) implements a single-shift version of the QZ method for finding the generalized eigenvalues of the complex matrix pair (A, B) which is in the generalized upper Hessenberg form. If the matrix pair (A, B) is not in the generalized upper Hessenberg form, then the function nag_zgghrd (f08wsc) should be called before invoking nag_zhgeqz (f08xsc).

This problem is mathematically equivalent to solving the matrix equation

$$\det(A - \lambda B) = 0.$$

Note that, to avoid underflow, overflow and other arithmetic problems, the generalized eigenvalues λ_j are never computed explicitly by this function but defined as ratios between two computed values, α_j and β_j:

$$\lambda_j = \alpha_j / \beta_j.$$

The arguments α_j, in general, are finite complex values and β_j are finite real non-negative values.

If desired, the matrix pair (A, B) may be reduced to generalized Schur form. That is, the transformed matrices A and B are upper triangular and the diagonal values of A and B provide α and β.

The argument **job** specifies two options. If **job** = Nag_Schur then the matrix pair (A, B) is simultaneously reduced to Schur form by applying one unitary transformation (usually called Q) on the left and another (usually called Z) on the right. That is,

$$A \leftarrow Q^{\mathrm{H}} A Z$$
$$B \leftarrow Q^{\mathrm{H}} B Z$$

If **job** = Nag_EigVals, then at each iteration the same transformations are computed but they are only applied to those parts of A and B which are needed to compute α and β. This option could be used if generalized eigenvalues are required but not generalized eigenvectors.

If **job** = Nag_Schur and **compq** = Nag_AccumulateQ or Nag_InitQ, and **compz** = Nag_AccumulateZ or Nag_InitZ, then the unitary transformations used to reduce the pair (A, B) are accumulated into the input arrays **q** and **z**. If generalized eigenvectors are required then **job** must be set to **job** = Nag_Schur and if left (right) generalized eigenvectors are to be computed then **compq** (**compz**) must be set to **compq** = Nag_AccumulateQ or Nag_InitQ rather than **compq** = Nag_NotQ.

If **compq** = Nag_InitQ, then eigenvectors are accumulated on the identity matrix and on exit the array **q** contains the left eigenvector matrix Q. However, if **compq** = Nag_AccumulateQ then the

transformations are accumulated in the user-supplied matrix Q_0 in array **q** on entry and thus on exit **q** contains the matrix product QQ_0. A similar convention is used for **compz**.

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

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

Moler C B and Stewart G W (1973) An algorithm for generalized matrix eigenproblems *SIAM J. Numer. Anal.* **10** 241–256

Stewart G W and Sun J-G (1990) *Matrix Perturbation Theory* Academic Press, London

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: specifies the operations to be performed on (A, B).

job = Nag_EigVals
The matrix pair (A, B) on exit might not be in the generalized Schur form.

job = Nag_Schur
The matrix pair (A, B) on exit will be in the generalized Schur form.

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

3: **compq** – Nag_ComputeQType *Input*

On entry: specifies the operations to be performed on Q:

compq = Nag_NotQ
The array **q** is unchanged.

compq = Nag_AccumulateQ
The left transformation Q is accumulated on the array **q**.

compq = Nag_InitQ
The array **q** is initialized to the identity matrix before the left transformation Q is accumulated in **q**.

Constraint: **compq** = Nag_NotQ, Nag_AccumulateQ or Nag_InitQ.

4: **compz** – Nag_ComputeZType *Input*

On entry: specifies the operations to be performed on Z.

compz = Nag_NotZ
The array **z** is unchanged.

compz = Nag_AccumulateZ
The right transformation Z is accumulated on the array **z**.

compz = Nag_InitZ
The array **z** is initialized to the identity matrix before the right transformation Z is accumulated in **z**.

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

5: **n** – Integer *Input*

On entry: n, the order of the matrices A, B, Q and Z.

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

6: **ilo** – Integer *Input*
7: **ihi** – Integer *Input*

On entry: the indices i_{lo} and i_{hi}, respectively which define the upper triangular parts of A. The submatrices $A(1 : i_{\mathrm{lo}} - 1, 1 : i_{\mathrm{lo}} - 1)$ and $A(i_{\mathrm{hi}} + 1 : n, i_{\mathrm{hi}} + 1 : n)$ are then upper triangular. These arguments are provided by nag_zggbal (f08wvc) if the matrix pair was previously balanced; otherwise, $\mathbf{ilo} = 1$ and $\mathbf{ihi} = \mathbf{n}$.

Constraints:

if $\mathbf{n} > 0$, $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq \mathbf{n}$;
if $\mathbf{n} = 0$, $\mathbf{ilo} = 1$ and $\mathbf{ihi} = 0$.

8: **a**[dim] – Complex *Input/Output*

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

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

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

On entry: the n by n upper Hessenberg matrix A. The elements below the first subdiagonal must be set to zero.

On exit: if **job** = Nag_Schur, the matrix pair (A, B) will be simultaneously reduced to generalized Schur form.

If **job** = Nag_EigVals, the 1 by 1 and 2 by 2 diagonal blocks of the matrix pair (A, B) will give generalized eigenvalues but the remaining elements will be irrelevant.

9: **pda** – Integer *Input*

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

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

10: **b**[dim] – Complex *Input/Output*

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

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

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

On entry: the n by n upper triangular matrix B. The elements below the diagonal must be zero.

On exit: if **job** = Nag_Schur, the matrix pair (A, B) will be simultaneously reduced to generalized Schur form.

If **job** = Nag_EigVals, the 1 by 1 and 2 by 2 diagonal blocks of the matrix pair (A, B) will give generalized eigenvalues but the remaining elements will be irrelevant.

11: **pdb** – Integer *Input*

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

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

12: **alpha**[**n**] – Complex *Output*

On exit: α_j, for $j = 1, 2, \ldots, n$.

13: **beta**[**n**] – Complex *Output*

On exit: β_j, for $j = 1, 2, \ldots, n$.

14: **q**[dim] – Complex *Input/Output*

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$ when **compq** = Nag_AccumulateQ or Nag_InitQ;
1 when **compq** = Nag_NotQ.

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

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

On entry: if **compq** = Nag_AccumulateQ, the matrix Q_0. The matrix Q_0 is usually the matrix Q returned by nag_zgghrd (f08wsc).

If **compq** = Nag_NotQ, **q** is not referenced.

On exit: if **compq** = Nag_AccumulateQ, **q** contains the matrix product QQ_0.

If **compq** = Nag_InitQ, **q** contains the transformation matrix Q.

15: **pdq** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor,

if **compq** = Nag_AccumulateQ or Nag_InitQ, **pdq** $\geq$ **n**;
if **compq** = Nag_NotQ, **pdq** $\geq$ 1.;

if **order** = Nag_RowMajor,

if **compq** = Nag_AccumulateQ or Nag_InitQ, **pdq** $\geq \max(1, \mathbf{n})$;
if **compq** = Nag_NotQ, **pdq** $\geq$ 1..

16: **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_AccumulateZ or Nag_InitZ;
1 when **compz** = Nag_NotZ.

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 entry: if **compz** = Nag_AccumulateZ, the matrix Z_0. The matrix Z_0 is usually the matrix Z returned by nag_zgghrd (f08wsc).

If **compz** = Nag_NotZ, **z** is not referenced.

On exit: if **compz** = Nag_AccumulateZ, **z** contains the matrix product ZZ_0.

If **compz** = Nag_InitZ, **z** contains the transformation matrix Z.

17: **pdz** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor,

if **compz** = Nag_AccumulateZ or Nag_InitZ, **pdz** $\geq$ **n**;
if **compz** = Nag_NotZ, **pdz** $\geq$ 1.;

if **order** = Nag_RowMajor,

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

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

On entry, **compq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: if **compq** = Nag_AccumulateQ or Nag_InitQ, **pdq** $\geq \max(1, \mathbf{n})$;
if **compq** = Nag_NotQ, **pdq** $\geq$ 1.

On entry, **compq** = ⟨*value*⟩, **pdq** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **compq** = Nag_AccumulateQ or Nag_InitQ, **pdq** $\geq$ **n**;
if **compq** = Nag_NotQ, **pdq** $\geq$ 1.

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

On entry, **compz** = ⟨*value*⟩, **pdz** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **compz** = Nag_AccumulateZ or Nag_InitZ, **pdz** $\geq$ **n**;
if **compz** = Nag_NotZ, **pdz** $\geq$ 1.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** $\geq$ 0.

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

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

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

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

NE_INT_2

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

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

NE_INT_3

On entry, **n** $= \langle value \rangle$, **ilo** $= \langle value \rangle$ and **ihi** $= \langle value \rangle$.
Constraint: if **n** > 0, $1 \leq$ **ilo** $\leq$ **ihi** $\leq$ **n**;
if **n** $= 0$, **ilo** $= 1$ and **ihi** $= 0$.

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 Library error has occurred.

NE_ITERATION_QZ

The QZ iteration did not converge and the matrix pair (A, B) is not in the generalized Schur form. The computed α_i and β_i should be correct for $i = \langle value \rangle, \ldots, \langle value \rangle$.

NE_SCHUR

The computation of shifts failed and the matrix pair (A, B) is not in the generalized Schur form. The computed α_i and β_i should be correct for $i = \langle value \rangle, \ldots, \langle value \rangle$.

7 Accuracy

Please consult Section 4.11 of the LAPACK Users' Guide (see Anderson *et al.* (1999)) and Chapter 6 of Stewart and Sun (1990), for more information.

8 Parallelism and Performance

nag_zhgeqz (f08xsc) is not threaded by NAG in any implementation.

nag_zhgeqz (f08xsc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

nag_zhgeqz (f08xsc) is the fifth step in the solution of the complex generalized eigenvalue problem and is called after nag_zgghrd (f08wsc).

The number of floating-point operations taken by this function is proportional to n^3.

The real analogue of this function is nag_dhgeqz (f08xec).

10 Example

This example computes the α and β arguments, which defines the generalized eigenvalues, of the matrix pair (A, B) given by

$$A = \begin{pmatrix} 1.0+3.0i & 1.0+4.0i & 1.0+5.0i & 1.0+6.0i \\ 2.0+2.0i & 4.0+3.0i & 8.0+4.0i & 16.0+5.0i \\ 3.0+1.0i & 9.0+2.0i & 27.0+3.0i & 81.0+4.0i \\ 4.0+0.0i & 16.0+1.0i & 64.0+2.0i & 256.0+3.0i \end{pmatrix}$$

and

$$B = \begin{pmatrix} 1.0+0.0i & 2.0+1.0i & 3.0+2.0i & 4.0+3.0i \\ 1.0+1.0i & 4.0+2.0i & 9.0+3.0i & 16.0+4.0i \\ 1.0+2.0i & 8.0+3.0i & 27.0+4.0i & 64.0+5.0i \\ 1.0+3.0i & 16.0+4.0i & 81.0+5.0i & 256.0+6.0i \end{pmatrix}.$$

This requires calls to five functions: nag_zggbal (f08wvc) to balance the matrix, nag_zgeqrf (f08asc) to perform the QR factorization of B, nag_zunmqr (f08auc) to apply Q to A, nag_zgghrd (f08wsc) to reduce the matrix pair to the generalized Hessenberg form and nag_zhgeqz (f08xsc) to compute the eigenvalues using the QZ algorithm.

10.1 Program Text

```
/* nag_zhgeqz (f08xsc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer       i, ihi, ilo, irows, j, n, pda, pdb;
  Integer       alpha_len, beta_len, scale_len, tau_len;
  Integer       exit_status = 0;

  NagError      fail;
  Nag_OrderType order;
  /* Arrays */
  Complex       *a = 0, *alpha = 0, *b = 0, *beta = 0, *q = 0, *tau = 0;
  Complex       *z = 0;
  Complex       e;
  double        *lscale = 0, *rscale = 0;

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

  INIT_FAIL(fail);

  printf("nag_zhgeqz (f08xsc) Example Program Results\n\n");
  /* Skip heading in data file */
  scanf("%*[^\n] ");
```

```
  scanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdb = n;
#else
  pda = n;
  pdb = n;
#endif
  alpha_len = n;
  beta_len = n;
  scale_len = n;
  tau_len = n;

  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, Complex)) ||
      !(alpha = NAG_ALLOC(alpha_len, Complex)) ||
      !(b = NAG_ALLOC(n * n, Complex)) ||
      !(beta = NAG_ALLOC(beta_len, Complex)) ||
      !(q = NAG_ALLOC(1 * 1, Complex)) ||
      !(tau = NAG_ALLOC(tau_len, Complex)) ||
      !(lscale = NAG_ALLOC(scale_len, double)) ||
      !(rscale = NAG_ALLOC(scale_len, double)) ||
      !(z = NAG_ALLOC(1 * 1, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* READ matrix A from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        scanf(" ( %lf, %lf ) ", &A(i, j).re, &A(i, j).im);
    }
  scanf("%*[^\n] ");

  /* READ matrix B from data file */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        scanf(" ( %lf, %lf ) ", &B(i, j).re, &B(i, j).im);
    }
  scanf("%*[^\n] ");
  /* Balance matrix pair (A,B) */
  /* nag_zggbal (f08wvc).
   * Balance a pair of complex general matrices
   */
  nag_zggbal(order, Nag_DoBoth, n, a, pda, b, pdb, &ilo, &ihi, lscale,
             rscale, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zggbal (f08wvc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Matrix A after balancing */
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                n, a, pda, Nag_BracketForm, "%7.4f",
                                "Matrix A after balancing",
                                Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
```

```
            fail.message);
    exit_status = 1;
    goto END;
  }
printf("\n");

/* Matrix B after balancing */
/* nag_gen_complx_mat_print_comp (x04dbc), see above. */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              n, b, pdb, Nag_BracketForm, "%7.4f",
                              "Matrix B after balancing",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                              0, 0, &fail);
if (fail.code != NE_NOERROR)
  {
    printf(
            "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }
printf("\n");

/* Reduce B to triangular form using QR */
irows = ihi + 1 - ilo;
/* nag_zgeqrf (f08asc).
 * QR factorization of complex general rectangular matrix
 */
nag_zgeqrf(order, irows, irows, &B(ilo, ilo), pdb, tau, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zgeqrf (f08asc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Apply the orthogonal transformation to matrix A */
/* nag_zunmqr (f08auc).
 * Apply unitary transformation determined by nag_zgeqrf
 * (f08asc) or nag_zgeqpf (f08bsc)
 */
nag_zunmqr(order, Nag_LeftSide, Nag_ConjTrans, irows, irows, irows,
           &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zunmqr (f08auc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute the generalized Hessenberg form of (A,B) */
/* nag_zgghrd (f08wsc).
 * Unitary reduction of a pair of complex general matrices
 * to generalized upper Hessenberg form
 */
nag_zgghrd(order, Nag_NotQ, Nag_NotZ, irows, 1, irows, &A(ilo, ilo),
           pda, &B(ilo, ilo), pdb, q, 1, z, 1, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zgghrd (f08wsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Matrix A in generalized Hessenberg form */
/* nag_gen_complx_mat_print_comp (x04dbc), see above. */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                              n, a, pda, Nag_BracketForm, "%7.3f",
                              "Matrix A in Hessenberg form",
```

```
                                Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\n");
  /* Matrix B in generalized Hessenberg form */
  /* nag_gen_complx_mat_print_comp (x04dbc), see above. */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                                n, b, pdb, Nag_BracketForm, "%7.3f",
                                "Matrix B is triangular",
                                Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute the generalized Schur form */
  /* nag_zhgeqz (f08xsc).
   * Eigenvalues and generalized Schur factorization of
   * complex generalized upper Hessenberg form reduced from a
   * pair of complex general matrices
   */
  nag_zhgeqz(order, Nag_EigVals, Nag_NotQ, Nag_NotZ, n, ilo, ihi, a,
             pda, b, pdb, alpha, beta, q, 1, z, 1, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhgeqz (f08xsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print the generalized eigenvalues */
  printf("\n Generalized eigenvalues\n");
  for (i = 0; i < n; ++i)
    {
      if (beta[i].re != 0.0 || beta[i].im != 0.0)
        {
          /* nag_complex_divide (a02cdc).
           * Quotient of two complex numbers
           */
          e = nag_complex_divide(alpha[i], beta[i]);

          printf(" %4ld     (%7.3f,%7.3f)\n", i+1, e.re, e.im);
        }
      else
        printf(" %4ld     Infinite eigenvalue\n", i+1);
    }
 END:
  NAG_FREE(a);
  NAG_FREE(alpha);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(lscale);
  NAG_FREE(q);
  NAG_FREE(rscale);
  NAG_FREE(tau);
  NAG_FREE(z);

  return exit_status;
```

```
}
```

10.2 Program Data

```
nag_zhgeqz (f08xsc) Example Program Data
   4                                                                :Value of N
(  1.00, 3.00)  (  1.00, 4.00)  (  1.00, 5.00)  (  1.00, 6.00)
(  2.00, 2.00)  (  4.00, 3.00)  (  8.00, 4.00)  ( 16.00, 5.00)
(  3.00, 1.00)  (  9.00, 2.00)  ( 27.00, 3.00)  ( 81.00, 4.00)
(  4.00, 0.00)  ( 16.00, 1.00)  ( 64.00, 2.00)  (256.00, 3.00)  :End of matrix A
(  1.00, 0.00)  (  2.00, 1.00)  (  3.00, 2.00)  (  4.00, 3.00)
(  1.00, 1.00)  (  4.00, 2.00)  (  9.00, 3.00)  ( 16.00, 4.00)
(  1.00, 2.00)  (  8.00, 3.00)  ( 27.00, 4.00)  ( 64.00, 5.00)
(  1.00, 3.00)  ( 16.00, 4.00)  ( 81.00, 5.00)  (256.00, 6.00)  :End of matrix B
```

10.3 Program Results

```
nag_zhgeqz (f08xsc) Example Program Results

 Matrix A after balancing
                     1                  2                  3                  4
 1  ( 1.0000, 3.0000)  ( 1.0000, 4.0000)  ( 0.1000, 0.5000)  ( 0.1000, 0.6000)
 2  ( 2.0000, 2.0000)  ( 4.0000, 3.0000)  ( 0.8000, 0.4000)  ( 1.6000, 0.5000)
 3  ( 0.3000, 0.1000)  ( 0.9000, 0.2000)  ( 0.2700, 0.0300)  ( 0.8100, 0.0400)
 4  ( 0.4000, 0.0000)  ( 1.6000, 0.1000)  ( 0.6400, 0.0200)  ( 2.5600, 0.0300)

 Matrix B after balancing
                     1                  2                  3                  4
 1  ( 1.0000, 0.0000)  ( 2.0000, 1.0000)  ( 0.3000, 0.2000)  ( 0.4000, 0.3000)
 2  ( 1.0000, 1.0000)  ( 4.0000, 2.0000)  ( 0.9000, 0.3000)  ( 1.6000, 0.4000)
 3  ( 0.1000, 0.2000)  ( 0.8000, 0.3000)  ( 0.2700, 0.0400)  ( 0.6400, 0.0500)
 4  ( 0.1000, 0.3000)  ( 1.6000, 0.4000)  ( 0.8100, 0.0500)  ( 2.5600, 0.0600)

 Matrix A in Hessenberg form
                     1                  2                  3                  4
 1  ( -2.868, -1.595)  ( -0.809, -0.328)  ( -4.900, -0.987)  ( -0.048,  1.163)
 2  ( -2.672,  0.595)  ( -0.790,  0.049)  ( -4.955, -0.163)  ( -0.439, -0.574)
 3  (  0.000,  0.000)  ( -0.098, -0.011)  ( -1.168, -0.137)  ( -1.756, -0.205)
 4  (  0.000,  0.000)  (  0.000,  0.000)  (  0.087,  0.004)  (  0.032,  0.001)

 Matrix B is triangular
                     1                  2                  3                  4
 1  ( -1.775,  0.000)  ( -0.721,  0.043)  ( -5.021,  1.190)  ( -0.145,  0.726)
 2  (  0.000,  0.000)  ( -0.218,  0.035)  ( -2.541, -0.146)  ( -0.823, -0.418)
 3  (  0.000,  0.000)  (  0.000,  0.000)  ( -1.396, -0.163)  ( -1.747, -0.204)
 4  (  0.000,  0.000)  (  0.000,  0.000)  (  0.000,  0.000)  ( -0.100, -0.004)

 Generalized eigenvalues
    1     ( -0.635,  1.653)
    2     (  0.458, -0.843)
    3     (  0.674, -0.050)
    4     (  0.493,  0.910)
```