NAG Library Function Document

nag_dhgeqz (f08xec)

1 Purpose

nag_dhgeqz (f08xec) implements the QZ method for finding generalized eigenvalues of the real 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_dhgeqz (Nag_OrderType order, Nag_JobType job,
     Nag_ComputeQType compq, Nag_ComputeZType compz, Integer n, Integer ilo,
     Integer ihi, double a[], Integer pda, double b[], Integer pdb,
     double alphar[], double alphai[], double beta[], double q[],
     Integer pdq, double z[], Integer pdz, NagError *fail)
```

3 Description

nag_dhgeqz (f08xec) implements a single-double-shift version of the QZ method for finding the generalized eigenvalues of the real 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_dgghrd (f08wec) should be called before invoking nag_dhgeqz (f08xec).

This problem is mathematically equivalent to solving the 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 matrix B is upper triangular and the transformed matrix A is block upper triangular, where the diagonal blocks are either 1 by 1 or 2 by 2. The 1 by 1 blocks provide generalized eigenvalues which are real and the 2 by 2 blocks give complex generalized eigenvalues.

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 orthogonal transformation (usually called Q) on the left and another (usually called Z) on the right. That is,

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

The 2 by 2 upper-triangular diagonal blocks of B corresponding to 2 by 2 blocks of **a** will be reduced to non-negative diagonal matrices. That is, if $\mathbf{A}(j+1, j)$ is nonzero, then $\mathbf{B}(j+1, j) = \mathbf{B}(j, j+1) = 0$ and $\mathbf{B}(j, j)$ and $\mathbf{B}(j+1, j+1)$ will be non-negative.

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 orthogonal 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 and not **compq** $\neq$ 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 on 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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_dggbal (f08whc) 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*] – double *Input/Output*

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

Where $\mathbf{A}(i, j)$ appears in this document, it refers to the array element

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*] – double *Input/Output*

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

Where $\mathbf{B}(i, j)$ appears in this document, it refers to the array element

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: **alphar**[**n**] – double *Output*

On exit: the real parts of α_j, for $j = 1, 2, \ldots, n$.

13: **alphai**[**n**] – double *Output*

On exit: the imaginary parts of α_j, for $j = 1, 2, \ldots, n$.

14: **beta**[**n**] – double *Output*

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

15: **q**[dim] – double *Input/Output*

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

max(1, **pdq** × **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

q[$(j-1) \times$ **pdq** $+ i - 1$] when **order** = Nag_ColMajor;
q[$(i-1) \times$ **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_dgghrd (f08wec).

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.

16: **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..

17: **z**[*dim*] – double *Input/Output*

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

max(1, **pdz** × **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

z[$(j-1) \times$ **pdz** $+ i - 1$] when **order** = Nag_ColMajor;
z[$(i-1) \times$ **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_dgghrd (f08wec).

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.

18: **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** ≥ **n**;
if **compz** = Nag_NotZ, **pdz** ≥ 1.;
if **order** = Nag_RowMajor,

if **compz** = Nag_AccumulateZ or Nag_InitZ, **pdz** ≥ max(1, **n**);
if **compz** = Nag_NotZ, **pdz** ≥ 1..

19: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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** ≥ max(1, **n**);
if **compq** = Nag_NotQ, **pdq** ≥ 1.

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

On entry, **compz** = ⟨*value*⟩, **pdz** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: if **compz** = Nag_AccumulateZ or Nag_InitZ, **pdz** ≥ max(1, **n**);
if **compz** = Nag_NotZ, **pdz** ≥ 1.

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

NE_INT

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **pda** $= \langle value \rangle$.
Constraint: **pda** > 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdq** $= \langle value \rangle$.
Constraint: **pdq** > 0.

On entry, **pdz** $= \langle value \rangle$.
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 error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

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_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

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_dhgeqz (f08xec) 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

nag_dhgeqz (f08xec) is the fifth step in the solution of the real generalized eigenvalue problem and is called after nag_dgghrd (f08wec).

The complex analogue of this function is nag_zhgeqz (f08xsc).

10 Example

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

$$A = \begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 2.0 & 4.0 & 8.0 & 16.0 & 32.0 \\ 3.0 & 9.0 & 27.0 & 81.0 & 243.0 \\ 4.0 & 16.0 & 64.0 & 256.0 & 1024.0 \\ 5.0 & 25.0 & 125.0 & 625.0 & 3125.0 \end{pmatrix}$$

$$B = \begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 \\ 1.0 & 4.0 & 9.0 & 16.0 & 25.0 \\ 1.0 & 8.0 & 27.0 & 64.0 & 125.0 \\ 1.0 & 16.0 & 81.0 & 256.0 & 625.0 \\ 1.0 & 32.0 & 243.0 & 1024.0 & 3125.0 \end{pmatrix}.$$

This requires calls to five functions: nag_dggbal (f08whc) to balance the matrix, nag_dgeqrf (f08aec) to perform the QR factorization of B, nag_dormqr (f08agc) to apply Q to A, nag_dgghrd (f08wec) to reduce the matrix pair to the generalized Hessenberg form and nag_dhgeqz (f08xec) to compute the eigenvalues using the QZ algorithm.

10.1 Program Text

```
/* nag_dhgeqz (f08xec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.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 */
  double *a = 0, *alphai = 0, *alphar = 0, *b = 0, *beta = 0;
  double *lscale = 0, *q = 0, *rscale = 0, *tau = 0, *z = 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_dhgeqz (f08xec) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
#endif
  pda = n;
  pdb = n;
  alpha_len = n;
  beta_len = n;
  scale_len = n;
  tau_len = n;

  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, double)) ||
      !(alphai = NAG_ALLOC(alpha_len, double)) ||
      !(alphar = NAG_ALLOC(alpha_len, double)) ||
      !(b = NAG_ALLOC(n * n, double)) ||
      !(beta = NAG_ALLOC(beta_len, double)) ||
      !(lscale = NAG_ALLOC(scale_len, double)) ||
      !(q = NAG_ALLOC(1 * 1, double)) ||
      !(rscale = NAG_ALLOC(scale_len, double)) ||
      !(tau = NAG_ALLOC(tau_len, double)) || !(z = NAG_ALLOC(1 * 1, double)))
  {
    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)
#ifdef _WIN32
      scanf_s("%lf", &A(i, j));
#else
      scanf("%lf", &A(i, j));
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* READ matrix B from data file */
  for (i = 1; i <= n; ++i) {
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s("%lf", &B(i, j));
```

```
#else
      scanf("%lf", &B(i, j));
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Balance matrix pair (A,B) */
  /* nag_dggbal (f08whc).
   * Balance a pair of real general matrices
   */
  nag_dggbal(order, Nag_DoBoth, n, a, pda, b, pdb, &ilo, &ihi, lscale,
             rscale, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dggbal (f08whc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Matrix A after balancing */
  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
                         pda, "Matrix A after balancing", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n");

  /* Matrix B after balancing */
  /* nag_gen_real_mat_print (x04cac), see above. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b,
                         pdb, "Matrix B after balancing", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n");

  /* Reduce B to triangular form using QR */
  irows = ihi + 1 - ilo;
  /* nag_dgeqrf (f08aec).
   * QR factorization of real general rectangular matrix
   */
  nag_dgeqrf(order, irows, irows, &B(ilo, ilo), pdb, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgeqrf (f08aec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Apply the orthogonal transformation to matrix A */
  /* nag_dormqr (f08agc).
   * Apply orthogonal transformation determined by nag_dgeqrf
   * (f08aec) or nag_dgeqpf (f08bec)
   */
  nag_dormqr(order, Nag_LeftSide, Nag_Trans, irows, irows, irows,
             &B(ilo, ilo), pdb, tau, &A(ilo, ilo), pda, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dormqr (f08agc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
```

```
  /* Compute the generalized Hessenberg form of (A,B) */
  /* nag_dgghd3 (f08wfc).
   * Orthogonal reduction of a pair of real general matrices
   * to generalized upper Hessenberg form
   */
  nag_dgghd3(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_dgghd3 (f08wfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Matrix A in generalized Hessenberg form */
  /* nag_gen_real_mat_print (x04cac), see above. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
                         pda, "Matrix A in Hessenberg form", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n");
  /* Matrix B in generalized Hessenberg form */
  /* nag_gen_real_mat_print (x04cac), see above. */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, b,
                         pdb, "Matrix B is triangular", 0, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Compute the generalized Schur form */
  /* nag_dhgeqz (f08xec).
   * Eigenvalues and generalized Schur factorization of real
   * generalized upper Hessenberg form reduced from a pair of
   * real general matrices
   */
  nag_dhgeqz(order, Nag_EigVals, Nag_NotQ, Nag_NotZ, n, ilo, ihi, a, pda,
             b, pdb, alphar, alphai, beta, q, 1, z, 1, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dhgeqz (f08xec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print the generalized eigenvalues */
  printf("\n Generalized eigenvalues\n");
  for (i = 1; i <= n; ++i) {
    if (beta[i - 1] != 0.0) {
      printf(" %4" NAG_IFMT "     (%7.3f,%7.3f)\n", i,
             alphar[i - 1] / beta[i - 1], alphai[i - 1] / beta[i - 1]);
    }
    else
      printf(" %4" NAG_IFMT "Eigenvalue is infinite\n", i);
  }
END:
  NAG_FREE(a);
  NAG_FREE(alphai);
  NAG_FREE(alphar);
  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_dhgeqz (f08xec) Example Program Data
   5                                                          :Value of N
1.00        1.00        1.00        1.00        1.00
2.00        4.00        8.00       16.00       32.00
3.00        9.00       27.00       81.00      243.00
4.00       16.00       64.00      256.00     1024.00
5.00       25.00      125.00      625.00     3125.00          :End of matrix A
1.00        2.00        3.00        4.00        5.00
1.00        4.00        9.00       16.00       25.00
1.00        8.00       27.00       64.00      125.00
1.00       16.00       81.00      256.00      625.00
1.00       32.00      243.00     1024.00     3125.00          :End of matrix B
```

10.3 Program Results

```
nag_dhgeqz (f08xec) Example Program Results

 Matrix A after balancing
           1          2          3          4          5
 1      1.0000     1.0000     0.1000     0.1000     0.1000
 2      2.0000     4.0000     0.8000     1.6000     3.2000
 3      0.3000     0.9000     0.2700     0.8100     2.4300
 4      0.4000     1.6000     0.6400     2.5600    10.2400
 5      0.5000     2.5000     1.2500     6.2500    31.2500

 Matrix B after balancing
           1          2          3          4          5
 1      1.0000     2.0000     0.3000     0.4000     0.5000
 2      1.0000     4.0000     0.9000     1.6000     2.5000
 3      0.1000     0.8000     0.2700     0.6400     1.2500
 4      0.1000     1.6000     0.8100     2.5600     6.2500
 5      0.1000     3.2000     2.4300    10.2400    31.2500

 Matrix A in Hessenberg form
           1          2          3          4          5
 1     -2.1898    -0.3181     2.0547     4.7371    -4.6249
 2     -0.8395    -0.0426     1.7132     7.5194   -17.1850
 3      0.0000    -0.2846    -1.0101    -7.5927    26.4499
 4      0.0000     0.0000     0.0376     1.4070    -3.3643
 5      0.0000     0.0000     0.0000     0.3813    -0.9937

 Matrix B is triangular
           1          2          3          4          5
 1     -1.4248    -0.3476     2.1175     5.5813    -3.9269
 2      0.0000    -0.0782     0.1189     8.0940   -15.2928
 3      0.0000     0.0000     1.0021   -10.9356    26.5971
 4      0.0000     0.0000     0.0000     0.5820    -0.0730
 5      0.0000     0.0000     0.0000     0.0000     0.5321

 Generalized eigenvalues
    1     ( -2.437,  0.000)
    2     (  0.607,  0.795)
    3     (  0.607, -0.795)
    4     (  1.000,  0.000)
    5     ( -0.410,  0.000)
```