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

```c
#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 \(\lambda_j\) are never computed explicitly by this function but defined as ratios between two computed values, \(\alpha_j\) and \(\beta_j\):

\[
\lambda_j = \frac{\alpha_j}{\beta_j}.
\]

The arguments \(\alpha_j\), in general, are finite complex values and \(\beta_j\) 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,

\[
A \leftarrow Q^T A Z \\
B \leftarrow Q^T B Z
\]

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 \(A(j+1,j)\) is nonzero, then \(B(j+1,j) = B(j,j+1) = 0\) and \(B(j,j)\) and \(B(j+1,j+1)\) 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 \(\alpha\) and \(\beta\). This option could be used if generalized eigenvectors 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 \texttt{compq} (\texttt{compz}) must be set to \texttt{compq} = \texttt{Nag\_AccumulateQ} or \texttt{Nag\_InitQ} and not \texttt{compq} \neq \texttt{Nag\_NotQ}.

If \texttt{compq} = \texttt{Nag\_InitQ}, then eigenvectors are accumulated on the identity matrix and on exit the array \texttt{q} contains the left eigenvector matrix \(Q\). However, if \texttt{compq} = \texttt{Nag\_AccumulateQ} then the transformations are accumulated on the user-supplied matrix \(Q_0\) in array \texttt{q} on entry and thus on exit \texttt{q} contains the matrix product \(QQ_0\). A similar convention is used for \texttt{compz}.

4 References


5 Arguments

1: \texttt{order} – \texttt{Nag\_OrderType} \hspace{1cm} \textit{Input}

\textit{On entry:} the \texttt{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 \texttt{order} = \texttt{Nag\_RowMajor}. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \texttt{order} = \texttt{Nag\_RowMajor} or \texttt{Nag\_ColMajor}.

2: \texttt{job} – \texttt{Nag\_JobType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies the operations to be performed on \((A, B)\).

\texttt{job} = \texttt{Nag\_EigVals}

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

\texttt{job} = \texttt{Nag\_Schur}

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

\textit{Constraint:} \texttt{job} = \texttt{Nag\_EigVals} or \texttt{Nag\_Schur}.

3: \texttt{compq} – \texttt{Nag\_ComputeQType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies the operations to be performed on \(Q\):

\texttt{compq} = \texttt{Nag\_NotQ}

The array \texttt{q} is unchanged.

\texttt{compq} = \texttt{Nag\_AccumulateQ}

The left transformation \(Q\) is accumulated on the array \texttt{q}.

\texttt{compq} = \texttt{Nag\_InitQ}

The array \texttt{q} is initialized to the identity matrix before the left transformation \(Q\) is accumulated in \texttt{q}.

\textit{Constraint:} \texttt{compq} = \texttt{Nag\_NotQ}, \texttt{Nag\_AccumulateQ} or \texttt{Nag\_InitQ}.
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:** $\text{compz} = \text{Nag_NotZ, Nag_AccumulateZ or Nag_InitZ}.$

**n** – Integer

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

**Constraint:** $n \geq 0$.

**ilo** – Integer

**ihi** – Integer

On entry: the indices $i_{lo}$ and $i_{hi}$, respectively which define the upper triangular parts of $A$. The submatrices $A(1 : i_{lo} - 1, 1 : i_{lo} - 1)$ and $A(i_{hi} + 1 : n, i_{hi} + 1 : n)$ are then upper triangular. These arguments are provided by nag_dggbal (f08whc) if the matrix pair was previously balanced; otherwise, $ilo = 1$ and $ihi = n$.

**Constraints:**

- if $n > 0$, $1 \leq ilo \leq ihi \leq n$;
- if $n = 0$, $ilo = 1$ and $ihi = 0$.

**a[\text{dim}]** – double

**Input/Output**

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

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

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

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

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

If $\text{job} = \text{Nag_EigVals}$, 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.

**pda** – Integer

**Input**

On entry: the stride separating row or column elements (depending on the value of $\text{order}$) in the array $a$.

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

**b[\text{dim}]** – double

**Input/Output**

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

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

- $b[(j - 1) \times \text{pdb} + i - 1]$ when $\text{order} = \text{Nag_ColMajor}$;
- $b[(i - 1) \times \text{pdb} + j - 1]$ when $\text{order} = \text{Nag_RowMajor}$.
On entry: the \( n \) by \( n \) upper triangular matrix \( B \). The elements below the diagonal must be zero.

On exit: if \( \text{job} = \text{Nag\_Schur} \), the matrix pair \( (A, B) \) will be simultaneously reduced to generalized Schur form.

If \( \text{job} = \text{Nag\_EigVals} \), 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: \( \text{pdb} \) – Integer

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

Constraint: \( \text{pdb} \geq \text{max}(1, n) \).

12: \( \text{alphar}[n] \) – double

On exit: the real parts of \( \alpha_j \), for \( j = 1, 2, \ldots, n \).

13: \( \text{alphai}[n] \) – double

On exit: the imaginary parts of \( \alpha_j \), for \( j = 1, 2, \ldots, n \).

14: \( \text{beta}[n] \) – double

On exit: \( \beta_j \), for \( j = 1, 2, \ldots, n \).

15: \( q[dim] \) – double

Input/Output

Note: the dimension, \( dim \), of the array \( q \) must be at least

\[
\text{max}(1, \text{pdq} \times n) \quad \text{when} \quad \text{compq} = \text{Nag\_AccumulateQ} \quad \text{or} \quad \text{Nag\_InitQ}; \\
1 \quad \text{when} \quad \text{compq} = \text{Nag\_NotQ}.
\]

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

\[
q[(j - 1) \times \text{pdq} + i - 1] \quad \text{when} \quad \text{order} = \text{Nag\_ColMajor}; \\
q[(i - 1) \times \text{pdq} + j - 1] \quad \text{when} \quad \text{order} = \text{Nag\_RowMajor}.
\]

On entry: if \( \text{compq} = \text{Nag\_AccumulateQ} \), the matrix \( Q_0 \). The matrix \( Q_0 \) is usually the matrix \( Q \) returned by \( \text{nag\_dgghrd (f08wec)} \).

If \( \text{compq} = \text{Nag\_NotQ} \), \( q \) is not referenced.

On exit: if \( \text{compq} = \text{Nag\_AccumulateQ} \), \( q \) contains the matrix product \( QQ_0 \).

If \( \text{compq} = \text{Nag\_InitQ} \), \( q \) contains the transformation matrix \( Q \).

16: \( \text{pdq} \) – Integer

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

Constraints:

\[
\begin{align*}
\text{if} \quad \text{order} = \text{Nag\_ColMajor}, \\
& \quad \text{if} \quad \text{compq} = \text{Nag\_AccumulateQ} \quad \text{or} \quad \text{Nag\_InitQ}, \quad \text{pdq} \geq n; \\
& \quad \text{if} \quad \text{compq} = \text{Nag\_NotQ}, \quad \text{pdq} \geq 1; \\
\text{if} \quad \text{order} = \text{Nag\_RowMajor}, \\
& \quad \text{if} \quad \text{compq} = \text{Nag\_AccumulateQ} \quad \text{or} \quad \text{Nag\_InitQ}, \quad \text{pdq} \geq \text{max}(1, n); \\
& \quad \text{if} \quad \text{compq} = \text{Nag\_NotQ}, \quad \text{pdq} \geq 1.
\end{align*}
\]
17: $z[dim]$ -- double  

**Input/Output**

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

\[
\max(1, pdz \times n) \text{ when } \text{compz} = \text{Nag_AccumulateZ or Nag_InitZ};
\]

1 when $\text{compz} = \text{Nag_NotZ}$.

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

\[
z[(j - 1) \times pdz + i - 1] \text{ when } order = \text{Nag_ColMajor};
\]

\[
z[(i - 1) \times pdz + j - 1] \text{ when } order = \text{Nag_RowMajor}.
\]

**On entry:** if $\text{compz} = \text{Nag_AccumulateZ}$, the matrix $Z_0$. The matrix $Z_0$ is usually the matrix $Z$ returned by nag_dgghrd (f08wec).

If $\text{compz} = \text{Nag_NotZ}$, $z$ is not referenced.

**On exit:** if $\text{compz} = \text{Nag_AccumulateZ}$, $z$ contains the matrix product $ZZ_0$.

If $\text{compz} = \text{Nag_InitZ}$, $z$ 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 $order = \text{Nag_ColMajor},$

- if $\text{compz} = \text{Nag_AccumulateZ or Nag_InitZ}$, $pdz \geq n$;
- if $\text{compz} = \text{Nag_NotZ}$, $pdz \geq 1$;

if $order = \text{Nag_RowMajor},$

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

19: $\text{fail} = \text{NagError *}$  

**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 $\langle\text{value}\rangle$ had an illegal value.

**NE_ENUM_INT_2**

On entry, $\text{compq} = \langle\text{value}\rangle$, $\text{pdq} = \langle\text{value}\rangle$, $n = \langle\text{value}\rangle$.

Constraint: if $\text{compq} = \text{Nag_AccumulateQ or Nag_InitQ}$, $pdq \geq \max(1, n)$;

if $\text{compq} = \text{Nag_NotQ}$, $pdq \geq 1$.

On entry, $\text{compq} = \langle\text{value}\rangle$, $\text{pdq} = \langle\text{value}\rangle$ and $n = \langle\text{value}\rangle$.

Constraint: if $\text{compq} = \text{Nag_AccumulateQ or Nag_InitQ}$, $pdq \geq \max(1, n)$;

if $\text{compq} = \text{Nag_NotQ}$, $pdq \geq 1$.

On entry, $\text{compz} = \langle\text{value}\rangle$, $\text{pdz} = \langle\text{value}\rangle$, $n = \langle\text{value}\rangle$.

Constraint: if $\text{compz} = \text{Nag_AccumulateZ or Nag_InitZ}$, $pdz \geq \max(1, n)$;

if $\text{compz} = \text{Nag_NotZ}$, $pdz \geq 1$. 

---

Mark 25
On entry, \( \text{compz} = \langle \text{value} \rangle, \) \( \text{pdz} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle. \)
Constraint: if \( \text{compz} = \text{Nag\_AccumulateZ} \) or \( \text{Nag\_InitZ} \), \( \text{pdz} \geq n; \)
if \( \text{compz} = \text{Nag\_NotZ}, \) \( \text{pdz} \geq 1. \)

**NE\_INT**

On entry, \( n = \langle \text{value} \rangle. \)
Constraint: \( n \geq 0. \)
On entry, \( \text{pda} = \langle \text{value} \rangle. \)
Constraint: \( \text{pda} > 0. \)
On entry, \( \text{pdb} = \langle \text{value} \rangle. \)
Constraint: \( \text{pdb} > 0. \)
On entry, \( \text{pdq} = \langle \text{value} \rangle. \)
Constraint: \( \text{pdq} > 0. \)
On entry, \( \text{pdz} = \langle \text{value} \rangle. \)
Constraint: \( \text{pdz} > 0. \)

**NE\_INT\_2**

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

**NE\_INT\_3**

On entry, \( n = \langle \text{value} \rangle, \) \( \text{ilo} = \langle \text{value} \rangle \) and \( \text{ihi} = \langle \text{value} \rangle. \)
Constraint: if \( n > 0, 1 \leq \text{ilo} \leq \text{ihi} \leq n; \)
if \( n = 0, \) \( \text{ilo} = 1 \) and \( \text{ihi} = 0. \)

**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 3.6.6 in the Essential Introduction 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 \( \alpha_i \) and \( \beta_i \) should be correct for \( i = \langle \text{value} \rangle, \ldots, \langle \text{value} \rangle. \)

**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\_SCHUR**

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

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) is not threaded by NAG in any implementation.

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 $\alpha$ and $\beta$ 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. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 7, 2001. */

#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nagf04.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;
alpalpha_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)
#endif
    scanf_s("%lf", &B(i, j));
#else
    scanf("%lf", &B(i, j));
#endif
}
#endif
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\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute the generalized Hessenberg form of (A,B) */
/* nag_dgghrd (f08wec).
 * Orthogonal reduction of a pair of real general matrices
 * to generalized upper Hessenberg form
 */
nag_dgghrd(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_dgghrd (f08wec).\n\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\n", fail.message);
    exit_status = 1;
    goto END;
}

/* 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\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\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the generalized eigenvalues */
printf("\nGeneralized 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(1scale);
NAG_FREE(q);
NAG_FREE(qscale);
NAG_FREE(tau);
NAG_FREE(z);
return exit_status;

10.2 Program Data

nag_dhgeqz (f08xec) Example Program Data

<table>
<thead>
<tr>
<th>Value of N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>1.00 1.00 1.00 1.00 1.00</td>
</tr>
<tr>
<td>2.00 4.00 8.00 16.00 32.00</td>
</tr>
<tr>
<td>3.00 9.00 27.00 81.00 243.00</td>
</tr>
<tr>
<td>4.00 16.00 64.00 256.00 1024.00</td>
</tr>
<tr>
<td>5.00 25.00 125.00 625.00 3125.00</td>
</tr>
</tbody>
</table>

: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 81.00 243.00 |
| 1.00 16.00 256.00 625.00 |
| 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

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 1.0000 & 1.0000 & 0.1000 & 0.1000 & 0.1000 \\
2 & 2.0000 & 4.0000 & 0.8000 & 1.6000 & 3.2000 \\
3 & 0.3000 & 0.9000 & 0.2700 & 0.8100 & 2.4300 \\
4 & 0.1000 & 1.6000 & 0.6400 & 2.5600 & 10.2400 \\
5 & 0.5000 & 2.5000 & 1.2500 & 6.2500 & 31.2500 \\
\end{array}
\]

Matrix B after balancing

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 1.0000 & 2.0000 & 0.3000 & 0.4000 & 0.5000 \\
2 & 1.0000 & 4.0000 & 0.9000 & 1.6000 & 2.5000 \\
3 & 0.1000 & 0.8000 & 0.2700 & 0.6400 & 1.2500 \\
4 & 0.1000 & 1.6000 & 0.8100 & 2.5600 & 6.2500 \\
5 & 0.1000 & 3.2000 & 2.4300 & 10.2400 & 31.2500 \\
\end{array}
\]

Matrix A in Hessenberg form

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & -2.1898 & -0.3181 & 2.0547 & 4.7371 & -4.6249 \\
2 & -0.8395 & -0.0426 & 1.7132 & 7.5194 & -17.1850 \\
3 & 0.0000 & -0.2846 & -1.0101 & -7.5927 & 26.4499 \\
4 & 0.0000 & 0.0000 & 0.0376 & 1.4070 & -3.3643 \\
5 & 0.0000 & 0.0000 & 0.0000 & 0.3813 & -0.9937 \\
\end{array}
\]

Matrix B is triangular

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & -1.4248 & -0.3476 & 2.1175 & 5.5813 & -3.9269 \\
\end{array}
\]
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)