NAG Library Function Document

nag_dgghrd (f08wec)

1 Purpose

nag_dgghrd (f08wec) reduces a pair of real matrices \((A, B)\), where \(B\) is upper triangular, to the generalized upper Hessenberg form using orthogonal transformations.

2 Specification

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

3 Description

nag_dgghrd (f08wec) is the third step in the solution of the real generalized eigenvalue problem

\[ Ax = \lambda Bx. \]

The (optional) first step balances the two matrices using nag_dggbal (f08wec). In the second step, matrix \(B\) is reduced to upper triangular form using the QR factorization function nag_dgeqrf (f08aec) and this orthogonal transformation \(Q\) is applied to matrix \(A\) by calling nag_dormqr (f08agc).

nag_dgghrd (f08wec) reduces a pair of real matrices \((A, B)\), where \(B\) is upper triangular, to the generalized upper Hessenberg form using orthogonal transformations. This two-sided transformation is of the form

\[
Q^T AZ = H \\
Q^T BZ = T
\]

where \(H\) is an upper Hessenberg matrix, \(T\) is an upper triangular matrix and \(Q\) and \(Z\) are orthogonal matrices determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices \(Q_1\) and \(Z_1\), so that

\[
Q_1 AZ_1^T = (Q_1 Q) H(Z_1 Z)^T, \\
Q_1 BZ_1^T = (Q_1 Q) T(Z_1 Z)^T.
\]

4 References


5 Arguments

1: \(\text{order} \quad \text{Nag\_OrderType} \quad \text{Input}\)

   \text{On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-}
   \text{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{compq} – \texttt{Nag\_ComputeQType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies the form of the computed orthogonal matrix \( Q \).

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

Do not compute \( Q \).

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

The orthogonal matrix \( Q \) is returned.

\texttt{compq} = \texttt{Nag\_UpdateSchur}

\( q \) must contain an orthogonal matrix \( Q_1 \), and the product \( Q_1Q \) is returned.

\textit{Constraint:} \texttt{compq} = \texttt{Nag\_NotQ}, \texttt{Nag\_InitQ} or \texttt{Nag\_UpdateSchur}.

3: \texttt{compz} – \texttt{Nag\_ComputeZType} \hspace{1cm} \textit{Input}

\textit{On entry:} specifies the form of the computed orthogonal matrix \( Z \).

\texttt{compz} = \texttt{Nag\_NotZ}

Do not compute \( Z \).

\texttt{compz} = \texttt{Nag\_InitZ}

The orthogonal matrix \( Z \) is returned.

\texttt{compz} = \texttt{Nag\_UpdateZ}

\( z \) must contain an orthogonal matrix \( Z_1 \), and the product \( Z_1Z \) is returned.

\textit{Constraint:} \texttt{compz} = \texttt{Nag\_NotZ}, \texttt{Nag\_UpdateZ} or \texttt{Nag\_InitZ}.

4: \texttt{n} – \texttt{Integer} \hspace{1cm} \textit{Input}

\textit{On entry:} \( n \), the order of the matrices \( A \) and \( B \).

\textit{Constraint:} \( n \geq 0 \).

5: \texttt{ilo} – \texttt{Integer} \hspace{1cm} \textit{Input}

6: \texttt{ihi} – \texttt{Integer} \hspace{1cm} \textit{Input}

\textit{On entry:} \( i_{lo} \) and \( i_{hi} \) as determined by a previous call to \texttt{nag\_dggbal} (f08wec). Otherwise, they should be set to 1 and \( n \), respectively.

\textit{Constraints:}

\begin{align*}
\text{if } & n > 0, \ 1 \leq ilo \leq ihi \leq n; \\
\text{if } & n = 0, \ i_{lo} = 1 \text{ and } i_{hi} = 0.
\end{align*}

7: \texttt{a[dim]} – \texttt{double} \hspace{1cm} \textit{Input/Output}

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

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

\begin{align*}
\text{a}[(j-1) \times pda + i - 1] & \text{ when } \texttt{order} = \texttt{Nag\_ColMajor}; \\
\text{a}[(i-1) \times pda + j - 1] & \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\end{align*}

\textit{On entry:} the matrix \( A \) of the matrix pair \((A, B)\). Usually, this is the matrix \( A \) returned by \texttt{nag\_dormqr} (f08agc).

\textit{On exit:} \texttt{a} is overwritten by the upper Hessenberg matrix \( H \).
On entry: the stride separating row or column elements (depending on the value of order) in the array a.

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

9: \( b[\text{dim}] \) – double

**Input/Output**

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

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

\[
\begin{align*}
  b[(j - 1) \times \text{pdb} + i - 1] & \quad \text{when order = Nag_ColMajor;} \\
  b[(i - 1) \times \text{pdb} + j - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

On entry: the upper triangular matrix \( B \) of the matrix pair \((A, B)\). Usually, this is the matrix \( B \) returned by the \( QR \) factorization function \text{nag_dgeqrf (f08aec)}.

On exit: \( b \) is overwritten by the upper triangular matrix \( T \).

10: \( \text{pdb} \) – Integer

**Input**

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

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

11: \( q[\text{dim}] \) – double

**Input/Output**

**Note:** the dimension, \( \text{dim} \), of the array \( q \) must be at least \( \max(1, \text{pdq} \times n) \) when \( \text{compq} = \text{Nag_InitQ} \) or \( \text{Nag_UpdateSchur} \);

1 when \( \text{compq} = \text{Nag_NotQ} \).

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

\[
\begin{align*}
  q[(j - 1) \times \text{pdq} + i - 1] & \quad \text{when order = Nag_ColMajor;} \\
  q[(i - 1) \times \text{pdq} + j - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

On entry: if \( \text{compq} = \text{Nag_UpdateSchur} \), \( q \) must contain an orthogonal matrix \( Q_1 \).

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

On exit: if \( \text{compq} = \text{Nag_InitQ} \), \( q \) contains the orthogonal matrix \( Q \).

If \( \text{compq} = \text{Nag_UpdateSchur} \), \( q \) is overwritten by \( Q_1 Q \).

12: \( \text{pdq} \) – Integer

**Input**

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

**Constraints:**

- if \( \text{compq} = \text{Nag_InitQ} \) or \( \text{Nag_UpdateSchur} \), \( \text{pdq} \geq \max(1, n) \);
- if \( \text{compq} = \text{Nag_NotQ} \), \( \text{pdq} \geq 1 \).

13: \( z[\text{dim}] \) – double

**Input/Output**

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

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

\[
\begin{align*}
  z[(j - 1) \times \text{pdz} + i - 1] & \quad \text{when order = Nag_ColMajor;} \\
  z[(i - 1) \times \text{pdz} + j - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

On entry: if \( \text{compz} = \text{Nag_UpdateZ} \), \( z \) must contain an orthogonal matrix \( Z_1 \).

If \( \text{compz} = \text{Nag_NotZ} \), \( z \) is not referenced.
On exit: if \( \text{compz} = \text{Nag_InitZ} \), \( z \) contains the orthogonal matrix \( Z \).

If \( \text{compz} = \text{Nag_UpdateZ} \), \( z \) is overwritten by \( Z_1 Z \).

14: \[ \text{pdz} \] – Integer

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

Constraints:

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

15: \[ \text{fail} \] – \text{NagError*}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\text{NE_ALLOC_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\text{NE_BAD_PARAM}

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

\text{NE_ENUM_INT_2}

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

Constraint: if \( \text{compq} = \text{Nag_InitQ} \) or \( \text{Nag_UpdateSchur} \), \( \text{pdq} \geq \max(1, n) \);
- if \( \text{compq} = \text{Nag_NotQ} \), \( \text{pdq} \geq 1 \).

On entry, \( \text{compz} = \langle \text{value} \rangle \), \( \text{pdz} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

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

\text{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: \( \text{pdz} > 0 \).

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

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.

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.

The reduction to the generalized Hessenberg form is implemented using orthogonal transformations
which are backward stable.

nag_dgghrd (f08wec) 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.

This function is usually followed by nag_dhgeqz (f08xec) which implements the $QZ$ algorithm for
computing generalized eigenvalues of a reduced pair of matrices.
The complex analogue of this function is nag_zgghrd (f08wsc).

See Section 10 in nag_dhgeqz (f08xec) and nag_dtgevc (f08ykc).