NAG Library Function Document

nag_zgghd3 (f08wtc)

1 Purpose

nag_zgghd3 (f08wtc) reduces a pair of complex matrices (A, B), where B is upper triangular, to the generalized upper Hessenberg form using unitary transformations.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
```

```
void nag_zgghd3 (Nag_OrderType order, Nag_ComputeQType compq,
     Nag_ComputeZType compz, Integer n, Integer ilo, Integer ihi,
     Complex a[], Integer pda, Complex b[], Integer pdb, Complex q[],
     Integer pdq, Complex z[], Integer pdz, NagError *fail)
```

3 Description

nag_zgghd3 (f08wtc) is usually the third step in the solution of the complex generalized eigenvalue problem

$$Ax = \lambda Bx.$$

The (optional) first step balances the two matrices using nag_zggbal (f08wvc). In the second step, matrix B is reduced to upper triangular form using the QR factorization function nag_zgeqrf (f08asc) and this unitary transformation Q is applied to matrix A by calling nag_zunmqr (f08auc). The driver, nag_zggev3 (f08wqc), solves the complex generalized eigenvalue problem by combining all the required steps including those just listed.

nag_zgghd3 (f08wtc) reduces a pair of complex matrices (A, B), where B is triangular, to the generalized upper Hessenberg form using unitary transformations. This two-sided transformation is of the form

$$\begin{gathered} Q^{\mathrm{H}} A Z = H, \\ Q^{\mathrm{H}} B Z = T \end{gathered}$$

where H is an upper Hessenberg matrix, T is an upper triangular matrix and Q and Z are unitary 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

$$\begin{gathered} Q_1 A Z_1^{\mathrm{H}} = (Q_1 Q) H (Z_1 Z)^{\mathrm{H}}, \\ Q_1 B Z_1^{\mathrm{H}} = (Q_1 Q) T (Z_1 Z)^{\mathrm{H}}. \end{gathered}$$

4 References

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th 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

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: **compq** – Nag_ComputeQType *Input*

On entry: specifies the form of the computed unitary matrix Q.

compq = Nag_NotQ
Do not compute Q.

compq = Nag_InitQ
The unitary matrix Q is returned.

compq = Nag_UpdateSchur
q must contain a unitary matrix Q_1, and the product Q_1Q is returned.

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

3: **compz** – Nag_ComputeZType *Input*

On entry: specifies the form of the computed unitary matrix Z.

compz = Nag_NotZ
Do not compute Z.

compz = Nag_UpdateZ
z must contain a unitary matrix Z_1, and the product Z_1Z is returned.

compz = Nag_InitZ
The unitary matrix Z is returned.

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

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

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

On entry: i_{lo} and i_{hi} as determined by a previous call to nag_zggbal (f08wvc). Otherwise, they should be set to 1 and n, respectively.

Constraints:

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

7: **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 matrix A of the matrix pair (A, B). Usually, this is the matrix A returned by nag_zunmqr (f08auc).

On exit: **a** is overwritten by the upper Hessenberg matrix H.

8: **pda** – Integer *Input*

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

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

9: **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

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

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 nag_zgeqrf (f08asc).

On exit: **b** is overwritten by the upper triangular matrix T.

10: **pdb** – Integer *Input*

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

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

11: **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_InitQ or Nag_UpdateSchur;
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_UpdateSchur, **q** must contain a unitary matrix Q_1.

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

On exit: if **compq** = Nag_InitQ, **q** contains the unitary matrix Q.

Iif **compq** = Nag_UpdateSchur, **q** is overwritten by $Q_1 Q$.

12: **pdq** – Integer *Input*

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

Constraints:

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

13: **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.

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_UpdateZ, **z** must contain a unitary matrix Z_1.

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

On exit: if **compz** = Nag_InitZ, **z** contains the unitary matrix Z.

If **compz** = Nag_UpdateZ, **z** is overwritten by $Z_1 Z$.

14: **pdz** – Integer *Input*

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

Constraints:

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

15: **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*⟩ and **n** = ⟨*value*⟩.
Constraint: if **compq** = Nag_InitQ or Nag_UpdateSchur, **pdq** $\geq \max(1, \mathbf{n})$;
if **compq** = Nag_NotQ, **pdq** ≥ 1.

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

NE_INT

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

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.

7 Accuracy

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

8 Parallelism and Performance

nag_zgghd3 (f08wtc) 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

This function is usually followed by nag_zhgeqz (f08xsc) which implements the QZ algorithm for computing generalized eigenvalues of a reduced pair of matrices.

The real analogue of this function is nag_dgghd3 (f08wfc).

10 Example

See Section 10 in nag_zhgeqz (f08xsc) and nag_ztgevc (f08yxc).