NAG C Library Function Document

nag_zunghr (f08ntc)

1 Purpose

nag_zunghr (f08ntc) generates the complex unitary matrix Q which was determined by nag_zgehrd (f08nsc) when reducing a complex general matrix A to Hessenberg form.

2 Specification

```
void nag_zunghr (Nag_OrderType order, Integer n, Integer ilo, Integer ihi,
     Complex a[], Integer pda, const Complex tau[], NagError *fail)
```

3 Description

nag_zunghr (f08ntc) is intended to be used following a call to nag_zgehrd (f08nsc), which reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation: $A = QHQ^H$. nag_zgehrd (f08nsc) represents the matrix Q as a product of $i_{hi} - i_{lo}$ elementary reflectors. Here i_{lo} and i_{hi} are values determined by nag_zgebal (f08nvc) when balancing the matrix; if the matrix has not been balanced, $i_{lo} = 1$ and $i_{hi} = n$.

This function may be used to generate Q explicitly as a square matrix. Q has the structure:

$$Q = \begin{pmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{pmatrix}$$

where Q_{22} occupies rows and columns i_{lo} to i_{hi}.

4 References

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

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **n** – Integer *Input*

On entry: n, the order of the matrix Q.

Constraint: **n** ≥ 0.

3: **ilo** – Integer *Input*
4: **ihi** – Integer *Input*

On entry: these **must** be the same parameters **ilo** and **ihi**, respectively, as supplied to nag_zgehrd (f08nsc).

Constraints:

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

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

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

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

If **order** = **Nag_ColMajor**, the (i, j)th element of the matrix A is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ and if **order** = **Nag_RowMajor**, the (i, j)th element of the matrix A is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zgehrd (f08nsc).

On exit: the n by n unitary matrix Q.

6: **pda** – Integer *Input*

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

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

7: **tau**[dim] – const Complex *Input*

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

On entry: further details of the elementary reflectors, as returned by nag_zgehrd (f08nsc).

8: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

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

On entry, **pda** = ⟨*value*⟩.
Constraint: $\mathbf{pda} > 0$.

NE_INT_2

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

NE_INT_3

On entry, **n** = ⟨*value*⟩, **ilo** = ⟨*value*⟩, **ihi** = ⟨*value*⟩.
Constraint: if $\mathbf{n} > 0$, $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq \mathbf{n}$;
if $\mathbf{n} = 0$, $\mathbf{ilo} = 1$ and $\mathbf{ihi} = 0$.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

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 consult NAG for assistance.

7 Accuracy

The computed matrix Q differs from an exactly unitary matrix by a matrix E such that

$$\|E\|_2 = O(\epsilon),$$

where ϵ is the ***machine precision***.

8 Further Comments

The total number of real floating-point operations is approximately $\frac{16}{3}q^3$, where $q = i_{hi} - i_{lo}$.

The real analogue of this function is nag_dorghr (f08nfc).

9 Example

To compute the Schur factorization of the matrix A, where

$$A = \begin{pmatrix} -3.97 - 5.04i & -4.11 + 3.70i & -0.34 + 1.01i & 1.29 - 0.86i \\ 0.34 - 1.50i & 1.52 - 0.43i & 1.88 - 5.38i & 3.36 + 0.65i \\ 3.31 - 3.85i & 2.50 + 3.45i & 0.88 - 1.08i & 0.64 - 1.48i \\ -1.10 + 0.82i & 1.81 - 1.59i & 3.25 + 1.33i & 1.57 - 3.44i \end{pmatrix}.$$

Here A is general and must first be reduced to Hessenberg form by nag_zgehrd (f08nsc). The program then calls nag_zunghr (f08ntc) to form Q, and passes this matrix to nag_zhseqr (f08psc) which computes the Schur factorization of A.

9.1 Program Text

```
/* nag_zunghr (f08ntc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
*/

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

int main(void)
{
/* Scalars */
  Integer  i, j, n, pda, pdz, tau_len, w_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_OrderType order;
/* Arrays */
  Complex *a=0, *tau=0, *w=0, *z=0;

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

  INIT_FAIL(fail);
  Vprintf("f08ntc Example Program Results\n\n");

/* Skip heading in data file */
  Vscanf("%*[^\n] ");
```

```
  Vscanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdz = n;
#else
  pda = n;
  pdz = n;
#endif
  tau_len = n - 1;
  w_len = n;

  /* Allocate memory */
  if ( !(a = NAG_ALLOC(n * n, Complex)) ||
       !(tau = NAG_ALLOC(tau_len, Complex)) ||
       !(w = NAG_ALLOC(w_len, Complex)) ||
       !(z = NAG_ALLOC(n * n, Complex)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

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

  /* Reduce A to upper Hessenberg form H = (Q**T)*A*Q */
  f08nsc(order, n, 1, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08nsc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Copy A into Z */
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        {
          Z(i,j).re = A(i,j).re;
          Z(i,j).im = A(i,j).im;
        }
    }

  /* Form Q explicitly, storing the result in Z */
  f08ntc(order, n, 1, n, z, pdz, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08ntc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Calculate the Schur factorization of H = Y*T*(Y**T) and form */
  /* Q*Y explicitly, storing the result in Z */

  /* Note that A = Z*T*(Z**T), where Z = Q*Y */
  f08psc(order, Nag_Schur, Nag_UpdateZ, n, 1, n, a, pda,
         w, z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08psc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
```

```
  /* Print Schur form */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         a, pda, Nag_BracketForm, "%7.4f", "Schur form",
         Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
         0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print Schur vectors */
  Vprintf("\n");
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
         z, pdz, Nag_BracketForm, "%7.4f",
         "Schur vectors of A", Nag_IntegerLabels, 0,
         Nag_IntegerLabels, 0, 80, 0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (a) NAG_FREE(a);
  if (tau) NAG_FREE(tau);
  if (w) NAG_FREE(w);
  if (z) NAG_FREE(z);

  return exit_status;
}
```

9.2 Program Data

```
f08ntc Example Program Data
  4                                                        :Value of N
 (-3.97,-5.04) (-4.11, 3.70) (-0.34, 1.01) ( 1.29,-0.86)
 ( 0.34,-1.50) ( 1.52,-0.43) ( 1.88,-5.38) ( 3.36, 0.65)
 ( 3.31,-3.85) ( 2.50, 3.45) ( 0.88,-1.08) ( 0.64,-1.48)
 (-1.10, 0.82) ( 1.81,-1.59) ( 3.25, 1.33) ( 1.57,-3.44)   :End of matrix A
```

9.3 Program Results

```
f08ntc Example Program Results

 Schur form
                   1                  2                  3                  4
 1  (-6.0004,-6.9998)  (-0.4701,-0.2119)  ( 0.0438, 0.5124)  (-0.9097,-0.0925)
 2  ( 0.0000, 0.0000)  (-5.0000, 2.0060)  ( 0.7150,-0.1028)  (-0.0580, 0.2575)
 3  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 7.9982,-0.9964)  (-0.2232,-1.0549)
 4  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 3.0023,-3.9998)

 Schur vectors of A
                   1                  2                  3                  4
 1  ( 0.8457, 0.0000)  (-0.3613, 0.1351)  (-0.1755, 0.2297)  ( 0.1099,-0.2007)
 2  (-0.0177, 0.3036)  (-0.3366, 0.4660)  ( 0.7228, 0.0000)  ( 0.0336, 0.2312)
 3  ( 0.0875, 0.3115)  ( 0.6311, 0.0000)  ( 0.2871, 0.4999)  ( 0.0944,-0.3947)
 4  (-0.0561,-0.2906)  (-0.1045,-0.3339)  ( 0.2476, 0.0195)  ( 0.8534, 0.0000)
```