NAG 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

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


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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

2:  

`n` – Integer

*Input*

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

Constraint: $n \geq 0$. 
3:  ilo – Integer  
    
    On entry: these must be the same arguments ilo and ihi, respectively, as supplied to nag_zgehrd (f08nsc).

    Constraints:
    if \( n > 0 \), \( 1 \leq \text{ilo} \leq \text{ihi} \leq n \);  
    if \( n = 0 \), \( \text{ilo} = 1 \) and \( \text{ihi} = 0 \).

4:  ihi – Integer  
    
    On entry: these must be the same arguments ilo and ihi, respectively, as supplied to nag_zgehrd (f08nsc).

    Constraints:
    if \( n > 0 \), \( 1 \leq \text{ilo} \leq \text{ihi} \leq n \);  
    if \( n = 0 \), \( \text{ilo} = 1 \) and \( \text{ihi} = 0 \).

5:  a[dim] – Complex  
    
    Note: the dimension, \( \text{dim} \), of the array \( a \) must be at least \( \max(1, \text{pda} \times n) \).

    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 \( Q \).

    If order = Nag_ColMajor, the \( (i,j) \)th element of the matrix is stored in \( a[(j-1) \times \text{pda} + i - 1] \).

    If order = Nag_RowMajor, the \( (i,j) \)th element of the matrix is stored in \( a[(i-1) \times \text{pda} + j - 1] \).

6:  pda – Integer  
    
    On entry: the stride separating row or column elements (depending on the value of order) in the array \( a \).

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

7:  tau[dim] – const Complex  
    
    Note: the dimension, \( \text{dim} \), of the array \( \tau \) must be at least \( \max(1, n - 1) \).

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

8:  fail – NagError *  
    
    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_INT  
On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( \text{pda} = \langle \text{value} \rangle \).

Constraint: \( \text{pda} > 0 \).

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, $n = \langle \text{value} \rangle$, $\text{ilo} = \langle \text{value} \rangle$ 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.

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.

7 Accuracy
The computed matrix $Q$ differs from an exactly unitary matrix by a matrix $E$ such that
$$\|E\|_2 = O(\epsilon),$$
where $\epsilon$ is the machine precision.

8 Parallelism and Performance
nag_zunghr (f08ntc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zunghr (f08ntc) 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
The total number of real floating-point operations is approximately $\frac{16}{3}q^3$, where $q = i_{\text{hi}} - i_{\text{lo}}$.
The real analogue of this function is nag_dorghr (f08nfc).

10 Example
This example computes 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 $A$. 
10.1 Program Text

/* nag_zunghr (f08ntc) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>
#include <nagx02.h>

int main(void)
{
    /* Scalars */
    Complex alpha, beta;
    double norm;
    Integer i, j, n, pda, pdc, pdd, pdz, tau_len, w_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *a = 0, *c = 0, *d = 0, *tau = 0, *w = 0, *z = 0;
    
    double norm;
    Integer i, j, n, pda, pdc, pdd, pdz, tau_len, w_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    
    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J - 1) * pda + I - 1]
    #define D(I, J) d[(J - 1) * pdd + 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 D(I, J) d[(I - 1) * pdd + J - 1]
    #define Z(I, J) z[(I - 1) * pdz + J - 1]
    order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);

    printf("nag_zunghr (f08ntc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifndef __WIN32
    scanf("%*[\n"");
    #else
    scanf("%*[\n"");
    #endif
    
    /* Skip heading in data file */
    #ifdef __WIN32
    scanf("%NAG_IFMT%*[\n"", &n);
    #else
    scanf("%NAG_IFMT%*[\n"", &n);
    #endif
    
    printf("nag_zunghr (f08ntc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef __WIN32
    scanf("%*[\n"");
    #else
    scanf("%*[\n"");
    #endif
    
    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdc = n;
    pdd = n;
    pdz = n;
    #else
    pda = n;
    pdc = n;
    pdd = n;
    pdz = n;
    #endif
    tau_len = n - 1;
    w_len = n;
/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, Complex)) ||
    !(c = NAG_ALLOC(n * n, Complex)) ||
    !(d = NAG_ALLOC(n * n, Complex)) ||
    !(tau = NAG_ALLOC(tau_len, Complex)) ||
    !(w = NAG_ALLOC(w_len, Complex)) ||
    !(z = NAG_ALLOC(n * n, Complex)))
{
    printf("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)
    {
        #ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
        #else
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
        #endif
    }
    #ifdef _WIN32
        scanf_s("%*[^
] ");
    #else
        scanf("%*[`\n ");
    #endif
}

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

/* nag_gen_complx_mat_print_comp (x04dbc): Print matrix A */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
    n, a, pda, Nag_BracketForm, "%7.4f",
    "Matrix A", Nag_IntegerLabels, 0,
    Nag_IntegerLabels, 0, 80, 0, 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Reduce A to upper Hessenberg form H = (Q**T)*A*Q */
/* nag_zgehrd (f08nsc). * Unitary reduction of complex general matrix to upper * Hessenberg form */
nag_zgehrd(order, n, 1, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zgehrd (f08nsc).\n\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 */  
/* nag_zunghr (f08ntc). */  
/* Generate unitary transformation matrix from reduction to */  
/* Hessenberg form determined by nag_zgehrd (f08nsc) */  
/* 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 */  
/* nag_zhseqr (f08psc). */  
/* Eigenvalues and Schur factorization of complex upper */  
/* Hessenberg matrix reduced from complex general matrix */  
/* nag_zgmm (f16zac): Compute A - Z*T*Z.H from the factorization of */  
/* A and store in matrix D*/  
alpha.re = 1.0;  
alpha.im = 0.0;  
beta.re = 0.0;  
beta.im = 0.0;  

if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zunghr (f08ntc).\n\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  

if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zhseqr (f08psc).\n\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  

if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zgmm (f16zac).\n\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  

if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zge_norm (f16uac).\n\n", fail.message);  
    exit_status = 1;  

if (norm>pow(x02ajc(),0.8))
{
    printf("%s
%s
","Norm of A-(Z*T*Z.H) is much greater than 0.,
    "Schur factorization has failed.");
}

END:
NAG_FREE(a);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(tau);
NAG_FREE(w);
NAG_FREE(z);
return exit_status;
}

10.2 Program Data

nag_zunghr (f08ntc) Example Program Data
   :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

10.3 Program Results

nag_zunghr (f08ntc) Example Program Results

Matrix A

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000, -5.0400</td>
<td>-4.1100, 3.7000</td>
<td>-0.3400, 1.0100</td>
<td>1.2900, -0.8600</td>
</tr>
<tr>
<td>0.3400, -1.5000</td>
<td>1.5200, -0.4300</td>
<td>1.8800, -5.3800</td>
<td>3.3600, 0.6500</td>
</tr>
<tr>
<td>3.3100, -3.8500</td>
<td>2.5000, 3.4500</td>
<td>0.8800, -1.0800</td>
<td>0.6400, -1.4800</td>
</tr>
<tr>
<td>-1.1000, 0.8200</td>
<td>1.8100, -1.5900</td>
<td>3.2500, 1.3300</td>
<td>1.5700, -3.4400</td>
</tr>
</tbody>
</table>