NAG Library Function Document

nag_zunmhr (f08nuc)

1 Purpose

nag_zunmhr (f08nuc) multiplies an arbitrary complex matrix $C$ by the complex unitary matrix $Q$ which was determined by nag_zgehrd (f08nsc) when reducing a complex general matrix to Hessenberg form.

2 Specification

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

void nag_zunmhr (Nag_OrderType order, Nag_SideType side,
    Nag_TransType trans, Integer m, Integer n, Integer ilo, Integer ihi,
    const Complex a[], Integer pda, const Complex tau[], Complex c[],
    Integer pdc, NagError *fail)
```

3 Description

nag_zunmhr (f08nuc) 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 form one of the matrix products

$$QC, Q^HC, CQ$$ or $$CQ^H,$$

overwriting the result on $C$ (which may be any complex rectangular matrix).

A common application of this function is to transform a matrix $V$ of eigenvectors of $H$ to the matrix $QV$ of eigenvectors of $A$.

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:  
   **side**  –  Nag_SideType  
   
   *Input*
   
   **On entry:** indicates how $Q$ or $Q^H$ is to be applied to $C$.
   
   side = Nag_LeftSide
   
   $Q$ or $Q^H$ is applied to $C$ from the left.
side = Nag_RightSide
    Q or \( Q^H \) is applied to \( C \) from the right.

Constraint: side = Nag_LeftSide or Nag_RightSide.

3:  trans – Nag_TransType  
    Input

On entry: indicates whether \( Q \) or \( Q^H \) is to be applied to \( C \).

trans = Nag_NoTrans
    \( Q \) is applied to \( C \).

trans = Nag_ConjTrans
    \( Q^H \) is applied to \( C \).

Constraint: trans = Nag_NoTrans or Nag_ConjTrans.

4:  m – Integer  
    Input

On entry: \( m \), the number of rows of the matrix \( C \); \( m \) is also the order of \( Q \) if side = Nag_LeftSide.

Constraint: \( m \geq 0 \).

5:  n – Integer  
    Input

On entry: \( n \), the number of columns of the matrix \( C \); \( n \) is also the order of \( Q \) if side = Nag_RightSide.

Constraint: \( n \geq 0 \).

6:  ilo – Integer  
7:  ihi – Integer  
    Input

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

Constraints:

    if side = Nag_LeftSide and \( m > 0 \), \( 1 \leq ilo \leq ihi \leq m \);
    if side = Nag_LeftSide and \( m = 0 \), \( ilo = 1 \) and \( ihi = 0 \);
    if side = Nag_RightSide and \( n > 0 \), \( 1 \leq ilo \leq ihi \leq n \);
    if side = Nag_RightSide and \( n = 0 \), \( ilo = 1 \) and \( ihi = 0 \).

8:  a[dim] – const Complex  
    Input

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

    \[ \max(1, pda \times m) \quad \text{when side = Nag_LeftSide} \]
    \[ \max(1, pda \times n) \quad \text{when side = Nag_RightSide} \]

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

9:  pda – Integer  
    Input

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

Constraints:

    if side = Nag_LeftSide, \( pda \geq \max(1, m) \);
    if side = Nag_RightSide, \( pda \geq \max(1, n) \).
10: \( \tau[\text{dim}] \) – const Complex

**Input**

Note: the dimension, \( \text{dim} \), of the array \( \tau \) must be at least
\[
\max(1, m - 1) \text{ when } \text{side} = \text{Nag LeftSide}; \\
\max(1, n - 1) \text{ when } \text{side} = \text{Nag RightSide}.
\]

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

11: \( c[\text{dim}] \) – Complex

**Input/Output**

Note: the dimension, \( \text{dim} \), of the array \( c \) must be at least
\[
\max(1, pdc \times n) \text{ when } \text{order} = \text{Nag ColMajor}; \\
\max(1, m \times pdc) \text{ when } \text{order} = \text{Nag RowMajor}.
\]

The \((i,j)\)th element of the matrix \( C \) is stored in
\[
c[(j - 1) \times pdc + i - 1] \text{ when } \text{order} = \text{Nag ColMajor}; \\
c[(i - 1) \times pdc + j - 1] \text{ when } \text{order} = \text{Nag RowMajor}.
\]

On entry: the \( m \) by \( n \) matrix \( C \).

On exit: \( c \) is overwritten by \( QC \) or \( Q^H C \) or \( CQ \) or \( CQ^H \) as specified by \text{side} and \text{trans}.

12: \( pdc \) – Integer

**Input**

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

Constraints:
\[
\text{if } \text{order} = \text{Nag ColMajor}, \ pdc \geq \max(1, m); \\
\text{if } \text{order} = \text{Nag RowMajor}, \ pdc \geq \max(1, n).
\]

13: \( \text{fail} \) – 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_3**

On entry, \( \text{side} = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \) and \( pda = \langle \text{value} \rangle \).
Constraint: if \( \text{side} = \text{Nag LeftSide} \), \( pda \geq \max(1, m) \);
if \( \text{side} = \text{Nag RightSide} \), \( pda \geq \max(1, n) \).

On entry, \( \text{side} = \langle \text{value} \rangle \), \( pda = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: if \( \text{side} = \text{Nag LeftSide} \), \( pda \geq \max(1, m) \);
if \( \text{side} = \text{Nag RightSide} \), \( pda \geq \max(1, n) \).

**NE_ENUM_INT_4**

On entry, \( \text{side} = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \), \( \text{ilo} = \langle \text{value} \rangle \) and \( \text{ihi} = \langle \text{value} \rangle \).
Constraint: if \( \text{side} = \text{Nag LeftSide} \) and \( m > 0 \), \( 1 \leq \text{ilo} \leq \text{ihi} \leq m \);
if \( \text{side} = \text{Nag LeftSide} \) and \( m = 0 \), \( \text{ilo} = 1 \) and \( \text{ihi} = 0 \);
if \( \text{side} = \text{Nag RightSide} \) and \( n > 0 \), \( 1 \leq \text{ilo} \leq \text{ihi} \leq n \);
if \( \text{side} = \text{Nag RightSide} \) and \( n = 0 \), \( \text{ilo} = 1 \) and \( \text{ihi} = 0 \).
NE_INT
On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m \geq 0 \).
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).
On entry, \( pda = \langle \text{value} \rangle \).
Constraint: \( pda > 0 \).
On entry, \( pdc = \langle \text{value} \rangle \).
Constraint: \( pdc > 0 \).

NE_INT_2
On entry, \( pdc = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( pdc \geq \max(1, m) \).
On entry, \( pdc = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdc \geq \max(1, n) \).

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 result differs from the exact result by a matrix \( E \) such that
\[
\| E \|_2 = O(\epsilon) \| C \|_2,
\]
where \( \epsilon \) is the machine precision.

8 Parallelism and Performance
\texttt{nag	extunderscore zumnhr (f08nuc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag	extunderscore zumnhr (f08nuc)} 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 \( 8nq^2 \) if \texttt{side = Nag	extunderscore LeftSide} and \( 8mq^2 \) if \texttt{side = Nag	extunderscore RightSide}, where \( q = i_{hi} - i_{lo} \).

The real analogue of this function is \texttt{nag	extunderscore dormhr (f08ngc)}.
10 Example

This example computes all the eigenvalues of the matrix \( A \), where

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

and those eigenvectors which correspond to eigenvalues \( \lambda \) such that \( \Re(\lambda) < 0 \). Here \( A \) is general and must first be reduced to upper Hessenberg form \( H \) by \nag_zgehrd (f08nsc). The program then calls \nag_zhseqr (f08psc) to compute the eigenvalues, and \nag_zhsein (f08pxc) to compute the required eigenvectors of \( H \) by inverse iteration. Finally \nag_zunmhr (f08nuc) is called to transform the eigenvectors of \( H \) back to eigenvectors of the original matrix \( A \).

10.1 Program Text

/* nag_zunmhr (f08nuc) Example Program. */
*  *
* Copyright 2014 Numerical Algorithms Group. *
*  *
* Mark 7, 2001. *
* Mark 7b revised, 2004. */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>
#include <naga02.h>

int main(void)
{
  /* Scalars */
  Integer i, j, m, n, pda, pdh, pdvl, pdvr, pdz;
  Integer tau_len, ifaill_len, select_len, w_len;
  Integer exit_status = 0;
  double thresh;
  NagError fail;
  Nag_OrderType order;
  /* Arrays */
  Complex *a = 0, *h = 0, *vl = 0, *vr = 0, *z = 0, *w = 0, *tau = 0;
  Integer *ifaill = 0, *ifailr = 0;
  Nag_Boolean *select = 0;
  #ifdef NAG_COLUMN_MAJOR
  #define A(I, J) a[(J - 1) * pda + I - 1]
  #define H(I, J) h[(J - 1) * pdh + I - 1]
  #define VR(I, J) vr[(J - 1) * pdvr + J - 1]
  #endif
  #ifdef NAG_ROW_MAJOR
  #define A(I, J) a[(I - 1) * pda + J - 1]
  #define H(I, J) h[(I - 1) * pdh + J - 1]
  #define VR(I, J) vr[(I - 1) * pdvr + J - 1]
  #endif
  order = Nag_RowMajor;
  #ifndef NAG_COLUMN_MAJOR
  order = Nag_ColumnMajor;
  #endif

  INIT_FAIL(fail);

  printf("nag_zunmhr (f08nuc) Example Program Results\n\n");
  /* Skip heading in data file */
  #ifdef _WIN32
  scanf_s("%*[\n ");
  #else
  scanf("%*[\n ");
  #endif
  #ifdef _WIN32
  #else
  #endif

  /* ... */
/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, Complex)) ||
    !(h = NAG_ALLOC(n * n, Complex)) ||
    !(vl = NAG_ALLOC(n * n, Complex)) ||
    !(vr = NAG_ALLOC(n * n, Complex)) ||
    !(z = NAG_ALLOC(1 * 1, Complex)) ||
    !(w = NAG_ALLOC(w_len, Complex)) ||
    !(ifaill = NAG_ALLOC(ifaill_len, Integer)) ||
    !(ifailr = NAG_ALLOC(ifaill_len, Integer)) ||
    !(select = NAG_ALLOC(select_len, Nag_Boolean)) ||
    !(tau = NAG_ALLOC(tau_len, 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
}

/* Reduce A to upper Hessenberg form */
/* 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%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Copy A to H */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
    {
        H(i, j).re = A(i, j).re;
\begin{verbatim}
H(i, j).im = A(i, j).im;
}

/* Calculate the eigenvalues of H (same as A) */
/* nag_zhseqr (f08psc). */
* Eigenvalues and Schur factorization of complex upper
* Hessenberg matrix reduced from complex general matrix
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zhseqr (f08psc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Calculate the eigenvalues */
if (i = 0; i < n; ++i)
    printf(" Eigenvalues\n");
for (i = 0; i < n; ++i)
    printf(" (%7.4f,%7.4f)", w[i].re, w[i].im);

/* Calculate the eigenvectors of H (as specified by SELECT), */
/* storing the result in VR */
/* nag_zhsein (f08pxc). */
* Selected right and/or left eigenvectors of complex upper
* Hessenberg matrix by inverse iteration
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zhsein (f08pxc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Calculate the eigenvectors of A = Q * VR */
/* nag_zunmhr (f08nuc). */
* Apply unitary transformation matrix from reduction to
* Hessenberg form determined by nag_zgehrd (f08nsc)
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zunmhr (f08nuc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Normalize the eigenvectors */
for(j=1; j<=m; j++)
    for(i=n; i>=1; i--)
        VR(i, j) = nag_complex_divide(VR(i, j), VR(1,j));

/* Print Eigenvectors */
fflush(stdout);

nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,

Mark 25
f08nuc.7
\end{verbatim}
m, vr, pdvr, Nag_BracketForm, "%7.4f",
"Contents of array VR", Nag_IntegerLabels, 0,
Nag_IntegerLabels, 0, 80, 0, 0, &fail);

if (fail.code != NE_NOERROR)
{
    printf(  
        "Error from nag_gen_complx_mat_print_comp (x04dbc). \n        %s\n",  
        fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(h);
NAG_FREE(vl);
NAG_FREE(vr);
NAG_FREE(z);
NAG_FREE(w);
NAG_FREE(ifaill);
NAG_FREE(ifailr);
NAG_FREE(select);
NAG_FREE(tau);
return exit_status;
}

10.2 Program Data

nag_zunmhr (f08nuc) Example Program Data

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(-3.97, -5.04)</td>
<td>(-4.11, 3.70)</td>
<td>(-0.34, 1.01)</td>
<td>(1.29, -0.86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0.34, -1.50)</td>
<td>(1.52, -0.43)</td>
<td>(1.88, -5.38)</td>
<td>(3.36, 0.65)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(3.31, -3.85)</td>
<td>(2.50, 3.45)</td>
<td>(0.88, -1.08)</td>
<td>(0.64, -1.48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(-1.10, 0.82)</td>
<td>(1.81, -1.59)</td>
<td>(3.25, 1.33)</td>
<td>(1.57, -3.44)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

:Value of N

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>(-6.0004, -6.9998)</td>
<td>(-5.0000, 2.0060)</td>
<td>(7.9982, -0.9964)</td>
<td>(3.0023, -3.9998)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

:End of matrix A

:Value of THRESH

10.3 Program Results

nag_zunmhr (f08nuc) Example Program Results

Eigenvalues

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(-6.0004, -6.9998)</td>
<td>(-5.0000, 2.0060)</td>
<td>(7.9982, -0.9964)</td>
<td>(3.0023, -3.9998)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contents of array VR

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1.0000, 0.0000)</td>
<td>(1.0000, 0.0000)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(-0.0210, 0.3590)</td>
<td>(1.1997, -0.6339)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(0.1035, 0.3683)</td>
<td>(-1.3192, -0.5912)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(-0.0664, -0.3436)</td>
<td>(-0.1319, 0.7904)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>