NAG C 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

```
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

$$QC,\ Q^HC,\ CQ \text{ or } CQ^H,$$

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

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: **side** – Nag_SideType *Input*

On entry: indicates how Q or Q^H is to be applied to C as follows:

if **side** = **Nag_LeftSide**, Q or Q^H is applied to C from the left;

if **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 as follows:

if **trans** = **Nag_NoTrans**, Q is applied to C;

if **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** ≥ 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** ≥ 0.

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

On entry: these **must** be the same parameters **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**$[dim]$ – Complex *Input/Output*

Note: the dimension, dim, of the array **a** must be at least

max(1, **pda** × **m**) when **side** = **Nag_LeftSide**;
max(1, **pda** × **n**) when **side** = **Nag_RightSide**.

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

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

On exit: used as internal workspace prior to being restored and hence is unchanged.

9: **pda** – Integer *Input*

On entry: the stride separating matrix 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**$[dim]$ – const Complex *Input*

Note: the dimension, dim, of the array **tau** must be at least max(1, **m** $- 1$) when **side** = **Nag_LeftSide** and at least max(1, **n** $- 1$) when **side** = **Nag_RightSide**.

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

11: **c**$[dim]$ – Complex *Input/Output*

Note: the dimension, dim, of the array **c** must be at least max(1, **pdc** × **n**) when **order** = **Nag_ColMajor** and at least max(1, **pdc** × **m**) when **order** = **Nag_RowMajor**.

If **order** $=$ **Nag_ColMajor**, the (i, j)th element of the matrix C is stored in **c**$[(j-1) \times$ **pdc** $+ i - 1]$ and if **order** $=$ **Nag_RowMajor**, the (i, j)th element of the matrix C is stored in **c**$[(i-1) \times$ **pdc** $+ j - 1]$.

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 **side** and **trans**.

12: **pdc** – Integer *Input*

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

Constraints:

if **order** $=$ **Nag_ColMajor**, **pdc** $\geq \max(1, \mathbf{m})$;
if **order** $=$ **Nag_RowMajor**, **pdc** $\geq \max(1, \mathbf{n})$.

13: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **m** = $\langle value \rangle$.
Constraint: **m** ≥ 0.

On entry, **n** = $\langle value \rangle$.
Constraint: **n** ≥ 0.

On entry, **pda** $= \langle value \rangle$.
Constraint: **pda** > 0.

On entry, **pdc** $= \langle value \rangle$.
Constraint: **pdc** > 0.

NE_INT_2

On entry, **pdc** $= \langle value \rangle$, **m** $= \langle value \rangle$.
Constraint: **pdc** $\geq \max(1, \mathbf{m})$.

On entry, **pdc** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdc** $\geq \max(1, \mathbf{n})$.

NE_ENUM_INT_3

On entry, **side** $= \langle value \rangle$, **m** $= \langle value \rangle$, **n** $= \langle value \rangle$, **pda** $= \langle value \rangle$.
Constraint: if **side** $=$ **Nag_LeftSide**, **pda** $\geq \max(1, \mathbf{m})$;
if **side** $=$ **Nag_RightSide**, **pda** $\geq \max(1, \mathbf{n})$.

NE_ENUM_INT_4

On entry, **side** $= \langle value \rangle$, **m** $= \langle value \rangle$, **n** $= \langle value \rangle$, **ilo** $= \langle value \rangle$, **ihi** $= \langle value \rangle$.
Constraint: 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** $= 0$.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 result differs from the exact result by a matrix E such that

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

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

8 Further Comments

The total number of real floating-point operations is approximately $8nq^2$ if **side** = **Nag_LeftSide** and $8mq^2$ if **side** = **Nag_RightSide**, where $q = i_{hi} - i_{lo}$.

The real analogue of this function is nag_dormhr (f08ngc).

9 Example

To compute 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 λ such that $\mathrm{Re}(\lambda) < 0$. 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.

9.1 Program Text

```
/* nag_zunmhr (f08nuc) 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, m, n, pda, pdh, pdvl, pdvr, pdz;
  Integer  tau_len, ifaill_len, ifailr_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;
  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]
  order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define H(I,J) h[(I-1)*pdh + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("f08nuc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);
#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdh = n;
  pdvl = n;
  pdvr = n;
  pdz = 1;
#else
  pda = n;
  pdh = n;
  pdvl = n;
  pdvr = n;
  pdz = 1;
#endif
  tau_len = n;
  w_len = n;
  ifaill_len = n;
  ifailr_len = n;
  select_len = n;

  /* 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, Boolean)) ||
       !(tau = NAG_ALLOC(tau_len, 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] ");
  Vscanf("%lf%*[^\n] ", &thresh);

  /* Reduce A to upper Hessenberg form */
  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 to H */
  for (i = 1; i <= n; ++i)
    {
```

```
      for (j = 1; j <= n; ++j)
        {
          H(i,j).re = A(i,j).re;
          H(i,j).im = A(i,j).im;
        }
    }

  /* Calculate the eigenvalues of H (same as A) */
  f08psc(order, Nag_EigVals, Nag_NotZ, n, 1, n, h, pdh, w,
         z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08psc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print eigenvalues */
  Vprintf(" Eigenvalues\n");
  for (i = 0; i < n; ++i)
    Vprintf(" (%7.4f,%7.4f)", w[i].re, w[i].im);
  Vprintf("\n");
  for (i = 0; i < n; ++i)
    select[i] = (w[i].re < thresh);
  /* Calculate the eigenvectors of H (as specified by SELECT), */
  /* storing the result in VR */
  f08pxc(order, Nag_RightSide, Nag_HSEQRSource, Nag_NoVec, select,
         n, a, pda, w, vl, pdvl, vr, pdvr, n, &m, ifaill,
         ifailr, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08pxc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Calculate the eigenvectors of A = Q * VR */
  f08nuc(order, Nag_LeftSide, Nag_NoTrans, n, m, 1, n, a, pda,
         tau, vr, pdvr, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08nuc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print Eigenvectors */
  Vprintf("\n");
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, 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)
    {
      Vprintf("Error from x04dbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  if (a) NAG_FREE(a);
  if (h) NAG_FREE(h);
  if (vl) NAG_FREE(vl);
  if (vr) NAG_FREE(vr);
  if (z) NAG_FREE(z);
  if (w) NAG_FREE(w);
  if (ifaill) NAG_FREE(ifaill);
  if (ifailr) NAG_FREE(ifailr);
  if (select) NAG_FREE(select);
  if (tau) NAG_FREE(tau);
  return exit_status;
}
```

9.2 Program Data

```
f08nuc 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
  0.0                                                         :Value of THRESH
```

9.3 Program Results

```
f08nuc Example Program Results

 Eigenvalues
 (-6.0004,-6.9998) (-5.0000, 2.0060) ( 7.9982,-0.9964) ( 3.0023,-3.9998)

 Contents of array VR
                  1                  2
 1  ( 1.0000,-0.0000)  ( 0.2613, 0.5284)
 2  (-0.0210, 0.3590)  ( 0.6485, 0.4683)
 3  ( 0.1035, 0.3683)  (-0.0323,-0.8516)
 4  (-0.0664,-0.3436)  (-0.4521, 0.1368)
```