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

```
#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: $A = QHQ^{\mathrm{H}}$. nag_zgehrd (f08nsc) represents the matrix Q as a product of $i_{\mathrm{hi}} - i_{\mathrm{lo}}$ 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, $i_{\mathrm{lo}} = 1$ and $i_{\mathrm{hi}} = n$.

This function may be used to form one of the matrix products

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

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 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: $\mathbf{m} \geq 0$.

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: $\mathbf{n} \geq 0$.

6: **ilo** – Integer *Input*
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 $\mathbf{m} > 0$, $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq \mathbf{m}$;
if **side** $=$ Nag_LeftSide and $\mathbf{m} = 0$, $\mathbf{ilo} = 1$ and $\mathbf{ihi} = 0$;
if **side** $=$ Nag_RightSide and $\mathbf{n} > 0$, $1 \leq \mathbf{ilo} \leq \mathbf{ihi} \leq \mathbf{n}$;
if **side** $=$ Nag_RightSide and $\mathbf{n} = 0$, $\mathbf{ilo} = 1$ and $\mathbf{ihi} = 0$.

8: **a**[*dim*] – const Complex *Input*

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

$\max(1, \mathbf{pda} \times \mathbf{m})$ when **side** $=$ Nag_LeftSide;
$\max(1, \mathbf{pda} \times \mathbf{n})$ 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, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **side** $=$ Nag_RightSide, $\mathbf{pda} \geq \max(1, \mathbf{n})$.

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

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

$\max(1, \mathbf{m} - 1)$ when **side** = Nag_LeftSide;
$\max(1, \mathbf{n} - 1)$ 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

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

The (i, j)th element of the matrix C is stored in

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

On entry: the m by n matrix C.

On exit: **c** is overwritten by QC or $Q^{\mathrm{H}}C$ or CQ or CQ^{H} as specified by **side** and **trans**.

12: **pdc** – Integer *Input*

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

Constraints:

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

13: **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.

NE_BAD_PARAM

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

NE_ENUM_INT_3

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

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

NE_ENUM_INT_4

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

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

On entry, $\mathbf{pdc} = \langle value \rangle$.
Constraint: $\mathbf{pdc} > 0$.

NE_INT_2

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

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

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.

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 Parallelism and Performance

nag_zunmhr (f08nuc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zunmhr (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 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 $\mathbf{side} = \text{Nag\_LeftSide}$ and $8mq^2$ if $\mathbf{side} = \text{Nag\_RightSide}$, where $q = i_{\text{hi}} - i_{\text{lo}}$.

The real analogue of this function is nag_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 λ such that $\text{Re}(\lambda) < 0$. 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 2001 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 + 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]
#define VR(I, J) vr[(I - 1) * pdvr + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_zunmhr (f08nuc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%*[^\n] ", &n);

  pda = n;
  pdh = n;
  pdvl = n;
  pdvr = n;
  pdz = 1;
  tau_len = n;
  w_len = n;
  ifaill_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, 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)
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    }
  scanf("%*[^\n] ");
  scanf("%lf%*[^\n] ", &thresh);

  /* 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;
          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
   */
  nag_zhseqr(order, Nag_EigVals, Nag_NotZ, n, 1, n, h, pdh, w,
             z, pdz, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zhseqr (f08psc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print eigenvalues */
  printf(" Eigenvalues\n");
  for (i = 0; i < n; ++i)
    printf(" (%7.4f,%7.4f)", w[i].re, w[i].im);
  printf("\n");
  for (i = 0; i < n; ++i)
    select[i] = w[i].re < thresh?Nag_TRUE:Nag_FALSE;
  /* 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
   */
  nag_zhsein(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)
    {
      printf("Error from nag_zhsein (f08pxc).\n%s\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)
   */
  nag_zunmhr(order, Nag_LeftSide, Nag_NoTrans, n, m, 1, n, a, pda,
             tau, vr, pdvr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zunmhr (f08nuc).\n%s\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 */
  printf("\n");
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(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)
    {
      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
  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
```

10.3 Program Results

```
nag_zunmhr (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)  ( 1.0000, 0.0000)
 2  (-0.0210, 0.3590)  ( 1.1997,-0.6339)
 3  ( 0.1035, 0.3683)  (-1.3192,-0.5912)
 4  (-0.0664,-0.3436)  (-0.1319, 0.7904)
```