# NAG Library Function Document

# nag_dormhr (f08ngc)

## 1 Purpose

nag_dormhr (f08ngc) multiplies an arbitrary real matrix $C$ by the real orthogonal matrix $Q$ which was determined by nag_dgehrd (f08nec) when reducing a real general matrix to Hessenberg form.

## 2 Specification

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

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

## 3 Description

nag_dormhr (f08ngc) is intended to be used following a call to nag_dgehrd (f08nec), which reduces a real general matrix $A$ to upper Hessenberg form $H$ by an orthogonal similarity transformation: $A = QHQ^{\mathrm{T}}$. nag_dgehrd (f08nec) represents the matrix $Q$ as a product of $i_{\mathrm{hi}} - i_{\mathrm{lo}}$ elementary reflectors. Here $i_{\mathrm{lo}}$ and $i_{\mathrm{hi}}$ are values determined by nag_dgebal (f08nhc) 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

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

overwriting the result on $C$ (which may be any real 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 $Q^{\mathrm{T}}$ is to be applied to $C$.

**side** = Nag_LeftSide
$Q$ or $Q^{\mathrm{T}}$ is applied to $C$ from the left.

**side** $=$ Nag_RightSide
$Q$ or $Q^{\mathrm{T}}$ 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 $Q^{\mathrm{T}}$ is to be applied to $C$.

**trans** $=$ Nag_NoTrans
$Q$ is applied to $C$.

**trans** $=$ Nag_Trans
$Q^{\mathrm{T}}$ is applied to $C$.

*Constraint*: **trans** $=$ Nag_NoTrans or Nag_Trans.

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 *Input*
7: **ihi** – Integer *Input*

*On entry*: these **must** be the same arguments **ilo** and **ihi**, respectively, as supplied to nag_dgehrd (f08nec).

*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 double *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_dgehrd (f08nec).

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

10: **tau**[*dim*] – const double *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_dgehrd (f08nec).

11: **c**[*dim*] – double *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 $Q^{\mathrm{T}}C$ or $CQ$ or $CQ^{\mathrm{T}}$ 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 $\epsilon$ is the ***machine precision***.

## 8 Parallelism and Performance

nag_dormhr (f08ngc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dormhr (f08ngc) 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 floating-point operations is approximately $2nq^2$ if **side** = Nag_LeftSide and $2mq^2$ if **side** = Nag_RightSide, where $q = i_{\mathrm{hi}} - i_{\mathrm{lo}}$.

The complex analogue of this function is nag_zunmhr (f08nuc).

## 10 Example

This example computes all the eigenvalues of the matrix $A$, where

$$A = \begin{pmatrix} 0.35 & 0.45 & -0.14 & -0.17 \\ 0.09 & 0.07 & -0.54 & 0.35 \\ -0.44 & -0.33 & -0.03 & 0.17 \\ 0.25 & -0.32 & -0.13 & 0.11 \end{pmatrix},$$

and those eigenvectors which correspond to eigenvalues $\lambda$ such that $\mathrm{Re}(\lambda) < 0$. Here $A$ is general and must first be reduced to upper Hessenberg form $H$ by nag_dgehrd (f08nec). The program then calls nag_dhseqr (f08pec) to compute the eigenvalues, and nag_dhsein (f08pkc) to compute the required eigenvectors of $H$ by inverse iteration. Finally nag_dormhr (f08ngc) is called to transform the eigenvectors of $H$ back to eigenvectors of the original matrix $A$.

## 10.1 Program Text

```
/* nag_dormhr (f08ngc) 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 <naga02.h>
#include <nagf08.h>
#include <nagx04.h>


int main(void)
{
  /* Scalars */
  Integer       i, j, k, m, n, pda, pdh, pdvl, pdvr, pdz;
  Integer       tau_len, ifaill_len, select_len, w_len;
  Integer       exit_status = 0;
  double        thresh;
  Complex       eig, eig1;
  /* Arrays */
  double        *a = 0, *h = 0, *vl = 0, *vr = 0, *z = 0, *wi = 0, *wr = 0;
  double        *tau = 0;
  Integer       *ifaill = 0, *ifailr = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  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_dormhr (f08ngc) 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, double)) ||
    !(h = NAG_ALLOC(n * n, double)) ||
    !(vl = NAG_ALLOC(n * n, double)) ||
    !(vr = NAG_ALLOC(n * n, double)) ||
    !(z = NAG_ALLOC(1 * 1, double)) ||
    !(wi = NAG_ALLOC(w_len, double)) ||
    !(wr = NAG_ALLOC(w_len, double)) ||
    !(ifaill = NAG_ALLOC(ifaill_len, Integer)) ||
    !(ifailr = NAG_ALLOC(ifaill_len, Integer)) ||
    !(select = NAG_ALLOC(select_len, Nag_Boolean)) ||
    !(tau = NAG_ALLOC(tau_len, double)))
  {
    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", &A(i, j));
scanf("%*[^\n]");
scanf("%lf%*[^\n]", &thresh);

/* Reduce A to upper Hessenberg form */
/* nag_dgehrd (f08nec).
 * Orthogonal reduction of real general matrix to upper
 * Hessenberg form
 */
nag_dgehrd(order, n, 1, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dgehrd (f08nec).\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) = A(i, j);

/* Calculate the eigenvalues of H (same as A) */
/* nag_dhseqr (f08pec).
 * Eigenvalues and Schur factorization of real upper
 * Hessenberg matrix reduced from real general matrix
 */
nag_dhseqr(order, Nag_EigVals, Nag_NotZ, n, 1, n, h, pdh, wr,
           wi, z, pdz, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dhseqr (f08pec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Print eigenvalues */
printf(" Eigenvalues\n");
for (i = 0; i < n; ++i)
  printf(" (%8.4f,%8.4f)\n", wr[i], wi[i]);
printf("\n");
for (i = 0; i < n; ++i)
  select[i] = wr[i] < thresh?Nag_TRUE:Nag_FALSE;
/* Calculate the eigenvectors of H (as specified by SELECT), */
/* storing the result in VR */
/* nag_dhsein (f08pkc).
 * Selected right and/or left eigenvectors of real upper
 * Hessenberg matrix by inverse iteration
 */
nag_dhsein(order, Nag_RightSide, Nag_HSEQRSource, Nag_NoVec, select,
           n, a, pda, wr, wi, vl, pdvl, vr, pdvr, n, &m, ifaill,
```

```
              ifailr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dhsein (f08pkc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Calculate the eigenvectors of A = Q * VR */
  /* nag_dormhr (f08ngc).
   * Apply orthogonal transformation matrix from reduction to
   * Hessenberg form determined by nag_dgehrd (f08nec)
   */
  nag_dormhr(order, Nag_LeftSide, Nag_NoTrans, n, m, 1, n, a, pda,
              tau, vr, pdvr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dormhr (f08ngc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Scale selected eigenvectors */
  j = 0;
  for(k=0; k<n; k++)
    {
      if (select[k]) {
        j++;
        if (wi[k]==0.0) {
          for(i=2; i<=n; i++) VR(i, j) = VR(i, j) / VR(1,j);
          VR(1,j) = 1.0;
        } else {
          eig1 = nag_complex(VR(1,j),VR(1,j+1));
          for(i=1; i<=n; i++) {
            eig = nag_complex(VR(i, j),VR(i, j+1));
            eig = nag_complex_divide(eig,eig1);
            VR(i,j) = eig.re;
            VR(i,j+1) = eig.im;
          }
          j++;
          k++;
        }
      }
    }
  /* Print Eigenvectors */
  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, vr,
                         pdvr, "Contents of array VR", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\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(wi);
  NAG_FREE(wr);
  NAG_FREE(ifaill);
  NAG_FREE(ifailr);
  NAG_FREE(select);
  NAG_FREE(tau);
  return exit_status;
```

```
}
```

## 10.2 Program Data

```
nag_dormhr (f08ngc) Example Program Data
  4                          :Value of N
  0.35   0.45  -0.14  -0.17
  0.09   0.07  -0.54   0.35
 -0.44  -0.33  -0.03   0.17
  0.25  -0.32  -0.13   0.11   :End of matrix A
  0.0                        :Value of THRESH
```

## 10.3 Program Results

```
nag_dormhr (f08ngc) Example Program Results

 Eigenvalues
 (  0.7995,  0.0000)
 ( -0.0994,  0.4008)
 ( -0.0994, -0.4008)
 ( -0.1007,  0.0000)

 Contents of array VR
             1          2          3
 1      1.0000     0.0000     1.0000
 2     -1.7779     0.3606     2.6491
 3     -0.9521     0.3411     4.7381
 4     -1.2785    -1.6841     5.7614
```