NAG C 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

```c
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: 
\[
A = QHQ^T.
\]

nag_dgehrd (f08nec) represents the matrix \( Q \) as a product of \( ihi - ilo \) elementary reflectors. Here \( ilo \) and \( ihi \) are values determined by nag_dgebal (f08nhc) when balancing the matrix; if the matrix has not been balanced, \( ilo = 1 \) and \( ihi = n \).

This function may be used to form one of the matrix products 
\[
QC, \quad Q^TC, \quad CQ \text{ or } CQ^T,
\]
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


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 \( Q^T \) is to be applied to \( C \) as follows:
   
   if `side = Nag_LeftSide`, \( Q \) or \( Q^T \) is applied to \( C \) from the left;
   
   if `side = Nag_RightSide`, \( Q \) or \( Q^T \) 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 \( Q^T \) is to be applied to \( C \) as follows:
if \(\text{trans} = \text{Nag}_\text{NoTrans}\), \(Q\) is applied to \(C\);  
if \(\text{trans} = \text{Nag}_\text{Trans}\), \(Q^T\) is applied to \(C\).

Constraint: \(\text{trans} = \text{Nag}_\text{NoTrans}\) or \(\text{Nag}_\text{Trans}\).

4: \(m\) – Integer \hspace{1cm} \text{Input}  
On entry: \(m\), the number of rows of the matrix \(C\); \(m\) is also the order of \(Q\) if \(\text{side} = \text{Nag}_\text{LeftSide}\).  
Constraint: \(m \geq 0\).

5: \(n\) – Integer \hspace{1cm} \text{Input}  
On entry: \(n\), the number of columns of the matrix \(C\); \(n\) is also the order of \(Q\) if \(\text{side} = \text{Nag}_\text{RightSide}\).  
Constraint: \(n \geq 0\).

6: \(ilo\) – Integer \hspace{1cm} \text{Input}  
7: \(ihi\) – Integer \hspace{1cm} \text{Input}  
On entry: these must be the same parameters \(ilo\) and \(ihi\), respectively, as supplied to \text{nag_dgehrd} (f08nec).  
Constraints:  
\[
\begin{align*}
\text{if} & \quad \text{side} = \text{Nag}_\text{LeftSide} \text{ and } m > 0, 1 \leq ilo \leq ihi \leq m; \\
\text{if} & \quad \text{side} = \text{Nag}_\text{LeftSide} \text{ and } m = 0, ilo = 1 \text{ and } ihi = 0; \\
\text{if} & \quad \text{side} = \text{Nag}_\text{RightSide} \text{ and } n > 0, 1 \leq ilo \leq ihi \leq n; \\
\text{if} & \quad \text{side} = \text{Nag}_\text{RightSide} \text{ and } n = 0, ilo = 1 \text{ and } ihi = 0.
\end{align*}
\]

8: \(a[\dim]\) – double \hspace{1cm} \text{Input/Output}  
Note: the dimension, \(\dim\), of the array \(a\) must be at least \(\max(1, \pda \times m)\) when \(\text{side} = \text{Nag}_\text{LeftSide}\); \(\max(1, \pda \times n)\) when \(\text{side} = \text{Nag}_\text{RightSide}\).  
If \(\text{order} = \text{Nag}_\text{ColMajor}\), the \((i, j)\)th element of the matrix \(A\) is stored in \(a[(j - 1) \times \pda + i - 1]\) and \(\text{if} \quad \text{order} = \text{Nag}_\text{RowMajor}\), 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 \text{nag_dgehrd} (f08nec).  
On exit: used as internal workspace prior to being restored and hence is unchanged.

9: \(\pda\) – Integer \hspace{1cm} \text{Input}  
On entry: the stride separating matrix row or column elements (depending on the value of \(\text{order}\)) in the array \(a\).  
Constraints:  
\[
\begin{align*}
\text{if} & \quad \text{side} = \text{Nag}_\text{LeftSide}, \pda \geq \max(1, m); \\
\text{if} & \quad \text{side} = \text{Nag}_\text{RightSide}, \pda \geq \max(1, n).
\end{align*}
\]

10: \(\tau[\dim]\) – const double \hspace{1cm} \text{Input}  
Note: the dimension, \(\dim\), of the array \(\tau\) must be at least \(\max(1, m - 1)\) when \(\text{side} = \text{Nag}_\text{LeftSide}\) and at least \(\max(1, n - 1)\) when \(\text{side} = \text{Nag}_\text{RightSide}\).  
On entry: further details of the elementary reflectors, as returned by \text{nag_dgehrd} (f08nec).

11: \(c[\dim]\) – double \hspace{1cm} \text{Input/Output}  
Note: the dimension, \(\dim\), of the array \(c\) must be at least \(\max(1, \pdc \times n)\) when \(\text{order} = \text{Nag}_\text{ColMajor}\) and at least \(\max(1, \pdc \times m)\) when \(\text{order} = \text{Nag}_\text{RowMajor}\).
If \( \text{order} = \text{Nag\_ColMajor} \), the \((i,j)\)th element of the matrix \( C \) is stored in \( c[(j-1) \times \text{pdc} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i,j)\)th element of the matrix \( C \) is stored in \( c[(i-1) \times \text{pdc} + j - 1] \).

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

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

12: \text{pdc} – Integer \hspace{1cm} \text{Input}

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

Constraints:
- If \( \text{order} = \text{Nag\_ColMajor} \), \text{pdc} \geq \max(1, m);
- If \( \text{order} = \text{Nag\_RowMajor} \), \text{pdc} \geq \max(1, n).

13: \text{fail} – NagError \hspace{1cm} \text{Output}

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

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, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \).
Constraint: \( \text{pdc} > 0 \).

NE\_INT\_2

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

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

NE\_ENUM\_INT\_3

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

NE\_ENUM\_INT\_4

On entry, \( \text{side} = \langle \text{value} \rangle \), \( m = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \), \( \text{ilo} = \langle \text{value} \rangle \), \( \text{ihi} = \langle \text{value} \rangle \).
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\_ALLOC\_FAIL

Memory allocation failed.

NE\_BAD\_PARAM

On entry, parameter \( \langle \text{value} \rangle \) 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 $\epsilon$ is the machine precision.

8 Further Comments

The total number of floating-point operations is approximately $2nq^2$ if $\text{side} = \text{Nag_LeftSide}$ and $2mq^2$ if $\text{side} = \text{Nag_RightSide}$, where $q = i_h - i_0$.

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

9 Example

To compute 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 $\text{Re}(\lambda) < 0$. 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 (f08nec) is called to transform the eigenvectors of $H$ back to eigenvectors of the original matrix $A$.

9.1 Program Text

/* nag_dormhr (f08ngc) Example Program. */
/* * Copyright 2001 Numerical Algorithms Group. */
/* * Mark 7, 2001. */
/*
#include <stdio.h>
#include <nag.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 */
    double *a=0, *h=0, *vl=0, *vr=0, *wi=0, *wr=0, *tau=0;
    Integer *ifaill=0, *ifailr=0;
    Boolean *select=0;
    #ifdef NAG_COLUMN_MAJOR

f08ngc
#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("f08ngc 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;
ifail_len = n;
ifailr_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)) ||
    !(ifail = NAG_ALLOC(ifail_len, Integer)) ||
    !(ifailr = NAG_ALLOC(ifailr_len, Integer)) ||
    !(select = NAG_ALLOC(select_len, Boolean)) ||
    !(tau = NAG_ALLOC(tau_len, double)) )
{
    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", &A(i,j));
}
Vscanf("%*[\n"]");
Vscanf("%lf%*[\n"]", &thresh);

/* Reduce A to upper Hessenberg form */
f08nec(order, n, 1, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nec.\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Copy A to H */
for (i = 1; i <= n; ++i)


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

/* Calculate the eigenvalues of H (same as A) */
f08pec(order, Nag_EigVals, Nag_NotZ, n, 1, n, h, pdh, wr, 
       wi, z, pdz, &fail);
if (fail.code != NE_NOERROR)
  { 
    Vprintf("Error from f08pec.\n\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Print eigenvalues */
Vprintf(" Eigenvalues\n");
for (i = 0; i < n; ++i)
  Vprintf("(%8.4f,%8.4f)\n", wr[i], wi[i]);
Vprintf("\n");
for (i = 0; i < n; ++i)
  select[i] = (wr[i] < thresh);
/* Calculate the eigenvectors of H (as specified by SELECT), */
/* storing the result in VR */
f08pkc(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)
  { 
    Vprintf("Error from f08pkc.\n\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Calculate the eigenvectors of A = Q * VR */
f08ngc(order, Nag_LeftSide, Nag_NoTrans, n, m, 1, n, a, pdvA, 
       tau, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
  { 
    Vprintf("Error from f08ngc.\n\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Print Eigenvectors */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, vr, pdvr, 
      "Contents of array VR", 0, &fail);
if (fail.code != NE_NOERROR)
  { 
    Vprintf("Error from x04cac.\n\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 (w1) NAG_FREE(w1);
if (wr) NAG_FREE(wr);
if (ifaill) NAG_FREE(ifaill);
if (ifaillr) NAG_FREE(ifailr);
if (select) NAG_FREE(select);
if (tau) NAG_FREE(tau);
return exit_status;
}
```
9.2 Program Data

f08ngc Example Program Data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.35</td>
<td>0.45</td>
<td>-0.14</td>
<td>-0.17</td>
</tr>
<tr>
<td>0.09</td>
<td>0.07</td>
<td>-0.54</td>
<td>0.35</td>
</tr>
<tr>
<td>-0.44</td>
<td>-0.33</td>
<td>-0.03</td>
<td>0.17</td>
</tr>
<tr>
<td>0.25</td>
<td>-0.32</td>
<td>-0.13</td>
<td>0.11</td>
</tr>
</tbody>
</table>

:Value of N

:End of matrix A

9.3 Program Results

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3881</td>
<td>0.0574</td>
</tr>
<tr>
<td>2</td>
<td>-0.7107</td>
<td>0.0380</td>
</tr>
<tr>
<td>3</td>
<td>-0.3891</td>
<td>0.0778</td>
</tr>
<tr>
<td>4</td>
<td>-0.3996</td>
<td>-0.7270</td>
</tr>
</tbody>
</table>

:Value of THRESH