NAG C Library Function Document

nag_dormbr (f08kgc)

1 Purpose

nag_dormbr (f08kgc) multiplies an arbitrary real matrix \( C \) by one of the real orthogonal matrices \( Q \) or \( P \) which were determined by nag_dgebrd (f08kec) when reducing a real matrix to bidiagonal form.

2 Specification

```c
void nag_dormbr (Nag_OrderType order, Nag_VectType vect, Nag_SideType side, Nag_TransType trans, Integer m, Integer n, Integer k, const double a[], Integer pda, const double tau[], double c[], Integer pdc, NagError *fail)
```

3 Description

nag_dormbr (f08kgc) is intended to be used after a call to nag_dgebrd (f08kec), which reduces a real rectangular matrix \( A \) to bidiagonal form \( B \) by an orthogonal transformation: \( A = QBPT \). nag_dgebrd (f08kec) represents the matrices \( Q \) and \( P^T \) as products of elementary reflectors.

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

\[
QC, Q^TC, CQ, CQ^T, PC, P^TC, CP \text{ or } CP^T,
\]

overwriting the result on \( C \) (which may be any real rectangular matrix).

4 References


5 Parameters

Note: in the descriptions below, \( r \) denotes the order of \( Q \) or \( P^T \): if \( \text{side} = \text{Nag_LeftSide} \), \( r = m \) and if \( \text{side} = \text{Nag_RightSide} \), \( r = n \).

1: \( \text{order} \) – Nag_OrderType

*Input*

On entry: the \( \text{order} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).

2: \( \text{vect} \) – Nag_VectType

*Input*

On entry: indicates whether \( Q \) or \( Q^T \) or \( P \) or \( P^T \) is to be applied to \( C \) as follows:

- if \( \text{vect} = \text{Nag_ApplyQ} \), \( Q \) or \( Q^T \) is applied to \( C \);
- if \( \text{vect} = \text{Nag_ApplyP} \), \( P \) or \( P^T \) is applied to \( C \).

Constraint: \( \text{vect} = \text{Nag_ApplyQ} \) or \( \text{Nag_ApplyP} \).

3: \( \text{side} \) – Nag.SideType

*Input*

On entry: indicates how \( Q \) or \( Q^T \) or \( P \) or \( P^T \) is to be applied to \( C \) as follows:
if side = Nag_LeftSide, $Q$ or $Q^T$ or $P$ or $P^T$ is applied to $C$ from the left;
if side = Nag_RightSide, $Q$ or $Q^T$ or $P$ or $P^T$ is applied to $C$ from the right.

Constraint: side = Nag_LeftSide or Nag_RightSide.

4: trans – Nag_TransType  
On entry: indicates whether $Q$ or $P$ or $QT$ or $PT$ is applied to $C$ as follows:
if trans = Nag_NoTrans, $Q$ or $P$ is applied to $C$;
if trans = Nag_Trans, $QT$ or $PT$ is applied to $C$.

Constraint: trans = Nag_NoTrans or Nag_Trans.

5: m – Integer  
On entry: $m_C$, the number of rows of the matrix $C$.

Constraint: $m \geq 0$.

6: n – Integer  
On entry: $n_C$, the number of columns of the matrix $C$.

Constraint: $n \geq 0$.

7: k – Integer  
On entry: if vect = Nag_ApplyQ, the number of columns in the original matrix $A$; if vect = Nag_ApplyP, the number of rows in the original matrix $A$.

Constraint: $k \geq 0$.

8: a[dim] – double  
Input/Output

Note: the dimension, $dim$, of the array $a$ must be at least
\[
\max(1, pda \times \max(1, \min(r, k))) \quad \text{when vect = Nag_ApplyQ and order = Nag_ColMajor;}
\]
\[
\max(1, pda \times r) \quad \text{when vect = Nag_ApplyQ and order = Nag_RowMajor;}
\]
\[
\max(1, pda \times r) \quad \text{when vect = Nag_ApplyP and order = Nag_ColMajor;}
\]
\[
\max(1, pda \times \min(r, k))) \quad \text{when vect = Nag_ApplyP and order = Nag_RowMajor.}
\]

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_dgebrd (f08kec).

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 order = Nag_ColMajor,
if vect = Nag_ApplyQ, $pda \geq \max(1, r)$;
if vect = Nag_ApplyP, $pda \geq \max(1, \min(r, k))$;
if order = Nag_RowMajor,
if vect = Nag_ApplyQ, $pda \geq \max(1, \min(r, k))$;
if vect = Nag_ApplyP, $pda \geq \max(1, r)$. 
On entry: further details of the elementary reflectors, as returned by nag_dgebrd (f08kec) in its parameter tauq if vect = Nag_ApplyQ, or in its parameter taup if vect = Nag_ApplyP.

On exit: c is overwritten by QC or QT C or CQ or CQT or PC or PT C or CP or CP T as specified by vect, side and trans.

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 ≥ max(1, m);
if order = Nag_RowMajor, pdc ≥ max(1, n).

fail – NagError *
Output
The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT
On entry, m = ⟨value⟩.
Constraint: m ≥ 0.

On entry, n = ⟨value⟩.
Constraint: n ≥ 0.

On entry, k = ⟨value⟩.
Constraint: k ≥ 0.

On entry, pda = ⟨value⟩.
Constraint: pda > 0.

On entry, pdc = ⟨value⟩.
Constraint: pdc > 0.

NE_INT_2
On entry, pdc = ⟨value⟩, m = ⟨value⟩.
Constraint: pdc ≥ max(1, m).

On entry, pdc = ⟨value⟩, n = ⟨value⟩.
Constraint: pdc ≥ max(1, n).

NE_ENUM_INT_2
On entry, vect = ⟨value⟩, k = ⟨value⟩, pda = ⟨value⟩.
Constraint: if vect = Nag_ApplyQ, pda ≥ max(1, r);
if vect = Nag_ApplyP, pda ≥ max(1, min(r, k)).
On entry, \( \text{vect} = \langle \text{value} \rangle, \ k = \langle \text{value} \rangle, \ \text{pda} = \langle \text{value} \rangle. \)

Constraint: if \( \text{vect} = \text{Nag}_\text{ApplyQ} \), \( \text{pda} \geq \max(1, \min(r, k)) \);
if \( \text{vect} = \text{Nag}_\text{ApplyP} \), \( \text{pda} \geq \max(1, r) \).

**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
- if \( \text{side} = \text{Nag}_\text{LeftSide} \) and \( m \geq k \), \( 2nk(2m - k) \);
- if \( \text{side} = \text{Nag}_\text{RightSide} \) and \( n \geq k \), \( 2nk(2n - k) \);
- if \( \text{side} = \text{Nag}_\text{LeftSide} \) and \( m < k \), \( 2m^2n \);
- if \( \text{side} = \text{Nag}_\text{RightSide} \) and \( n < k \), \( 2mn^2 \),

where \( k \) is the value of the parameter \( k \).

The complex analogue of this function is \text{nag_zunmbr} (f08kuc).

### 9 Example

For this function two examples are presented. Both illustrate how the reduction to bidiagonal form of a matrix \( A \) may be preceded by a \( QR \) or \( LQ \) factorization of \( A \).

In the first example, \( m > n \), and

\[
A = \begin{pmatrix}
-0.57 & -1.28 & -0.39 & 0.25 \\
-1.93 & 1.08 & -0.31 & -2.14 \\
2.30 & 0.24 & 0.40 & -0.35 \\
-1.93 & 0.64 & -0.66 & 0.08 \\
0.15 & 0.30 & 0.15 & -2.13 \\
-0.02 & 1.03 & -1.43 & 0.50
\end{pmatrix}.
\]

The function first performs a \( QR \) factorization of \( A \) as \( A = Q_a R \) and then reduces the factor \( R \) to bidiagonal form \( B : R = Q_b B P^T \). Finally it forms \( Q_a \) and calls \text{nag_dormbr} (f08kge) to form \( Q = Q_a Q_b \).

In the second example, \( m < n \), and

\[
A = \begin{pmatrix}
-5.42 & 3.28 & -3.68 & 0.27 & 2.06 & 0.46 \\
-1.65 & -3.40 & -3.20 & -1.03 & -4.06 & -0.01 \\
-0.37 & 2.35 & 1.90 & 4.31 & -1.76 & 1.13 \\
-3.15 & -0.11 & 1.99 & -2.70 & 0.26 & 4.50
\end{pmatrix}.
\]

The function first performs an \( LQ \) factorization of \( A \) as \( A = L P_a^T \) and then reduces the factor \( L \) to
bidiagonal form $B: L = QBP_B^T$. Finally it forms $P_B^T$ and calls nag_dormbr (f08kgc) to form $P^T = P_B^T P_a^T$.

9.1 Program Text

```c
/* nag_dormbr (f08kgc) 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, ic, j, m, n, pda, pdpt, pdu;
    Integer d_len, e_len, tau_len, tauq_len, taup_len;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *a=0, *d=0, *e=0, *pt=0, *tau=0, *taup=0, *tauq=0, *u=0;
    #ifdef NAG_COLUMN_MAJOR
    #define A(I,J) a[(J-1)*pda+I-1]
    #define U(I,J) u[(J-1)*pdu+I-1]
    #define PT(I,J) pt[(J-1)*pdpt+I-1]
    order = Nag_ColMajor;
    #else
    #define A(I,J) a[(I-1)*pda+J-1]
    #define U(I,J) u[(I-1)*pdu+J-1]
    #define PT(I,J) pt[(I-1)*pdpt+J-1]
    order = Nag_RowMajor;
    #endif
    INIT_FAIL(fail);
    Vprintf("f08kgc Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%[*\n"]);
    for (ic = 1; ic <= 2; ++ic)
    { Vscanf("%ld%ld%[*\n"]", &m, &n);
        #ifdef NAG_COLUMN_MAJOR
        pda = m;
        pdu = m;
        pdpt = n;
        taup_len = n;
        tauq_len = n;
        tau_len = n;
        d_len = n;
        e_len = n-1;
        #else
        pda = n;
        pdu = m;
        pdpt = n;
        taup_len = n;
        tauq_len = n;
        tau_len = n;
        d_len = n;
        e_len = n-1;
        #endif
        /* Allocate memory */
```
if ( !(a = NAG_ALLOC(m * n, double)) ||
!(d = NAG_ALLOC(d_len, double)) ||
!(e = NAG_ALLOC(e_len, double)) ||
!(pt = NAG_ALLOC(n * n, double)) ||
!(tau = NAG_ALLOC(tau_len, double)) ||
!(taup = NAG_ALLOC(taup_len, double)) ||
!(tauq = NAG_ALLOC(tauq_len, double)) ||
!(u = NAG_ALLOC(m * m, double)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read A from data file */
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= n; ++j)
        Vscanf("%lf", &A(i,j));
    Vscanf("%*[\n\"]");
    if (m >= n)
    {
        /* Compute the QR factorization of A */
        f08aec(order, m, n, a, pda, tau, &fail);
        if (fail.code != NE_NOERROR)
        {
            Vprintf("Error from f08aec.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
        }
        /* Copy A to U */
        for (i = 1; i <= m; ++i)
        {
            for (j = 1; j <= MIN(i,n); ++j)
                U(i,j) = A(i,j);
        }
        /* Form Q explicitly, storing the result in U */
        f08afc(order, m, m, n, u, pdu, tau, &fail);
        if (fail.code != NE_NOERROR)
        {
            Vprintf("order=%d\n", order);
            Vprintf("Error from f08afc.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
        }
        /* Copy R to PT (used as workspace) */
        for (i = 1; i <= n; ++i)
        {
            for (j = i; j <= n; ++j)
                PT(i,j) = A(i,j);
        }
        /* Set the strictly lower triangular part of R to zero */
        for (i = 2; i <= n; ++i)
        {
            for (j = 1; j <= MIN(i-1,n-1); ++j)
                PT(i,j) = 0.0;
        }
        /* Bidiagonalize R */
        f08kec(order, n, n, pt, pdpt, d, e, tauq, taup, &fail);
        if (fail.code != NE_NOERROR)
        {
            Vprintf("Error from f08kec.\n%s\n", fail.message);
            exit_status = 1;
            goto END;
        }
        /* Update Q, storing the result in U */
        f08kgc(order, Nag_FormQ, Nag_RightSide, Nag_NoTrans,
               m, n, pt, pdpt, tauq, u, pdu, &fail);
        if (fail.code != NE_NOERROR)
        {
            Vprintf("Error from f08kgc.\n%s\n", fail.message);
        }
    }
}
exit_status = 1;
goto END;
}

/* Print bidiagonal form and matrix Q */
Vprintf("\nExample 1: bidiagonal matrix B\nDiagonal\n");
for (i = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
Vprintf("\nSuper-diagonal\n");
for (i = 1; i <= n - 1; ++i)
    Vprintf("%8.4f%s", e[i-1], i%8 == 0 ?"\n":" ");
Vprintf("\n\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
m, n, u, pdu, "Example 1: matrix Q", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04cac.\n%s\n", fail.message);
    exit_status = 1;
goto END;
}

else
{
    /* Compute the LQ factorization of A */
f08ahc(order, m, n, a, pda, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08ahc.\n%s\n", fail.message);
        exit_status = 1;
goto END;
    }
    /* Copy A to PT */
    for (i = 1; i <= m; ++i)
    {
        for (j = i; j <= n; ++j)
            PT(i,j) = A(i,j);
    }
    /* Form Q explicitly, storing the result in PT */
f08ajc(order, n, n, m, pt, pdpt, tau, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08ajc.\n%s\n", fail.message);
        exit_status = 1;
goto END;
    }
    /* Copy L to U (used as workspace) */
    for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= i; ++j)
            U(i,j) = A(i,j);
    }
    /* Set the strictly upper triangular part of L to zero */
    for (i = 1; i <= m-1; ++i)
    {
        for (j = i+1; j <= m; ++j)
            U(i,j) = 0.0;
    }
    /* Bidiagonalize L */
f08kec(order, m, m, u, pdu, d, e, tauq, taup, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08kec.\n%s\n", fail.message);
        exit_status = 1;
goto END;
    }
    /* Update P**T, storing the result in PT */
f08kgc(order, Nag_FormP, Nag_LeftSide, Nag_Trans,
m, n, m, u, pdu, taup, pt, pdpt, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from f08kgc.\n%s\n", fail.message);
        exit_status = 1;
    }
}
/* Print bidiagonal form and matrix P**T */
Vprintf("\nExample 2: bidiagonal matrix B\n\%s\n", "Diagonal");
for (i = 1; i <= m; ++i)
  Vprintf("%8.4f%s", d[i-1], i%8==0 ?"\n":" ");
Vprintf("\nSuper-diagonal\n");
for (i = 1; i <= m - 1; ++i)
  Vprintf("%8.4f%s", e[i-1], i%8==0 ?"\n":" ");
Vprintf("\n\n");
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
  m, n, pt, pdpt, "Example 2: matrix P**T", 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 (d) NAG_FREE(d);
if (e) NAG_FREE(e);
if (pt) NAG_FREE(pt);
if (tau) NAG_FREE(tau);
if (taup) NAG_FREE(taup);
if (tauq) NAG_FREE(tauq);
if (u) NAG_FREE(u);
return exit_status;
}

9.2 Program Data
f08kgc Example Program Data

6 4 :Values of M and N, Example 1
-0.57 -1.28 -0.39 0.25
-1.93 1.08 -0.31 -2.14
2.30 0.24 0.40 -0.35
-1.93 0.64 -0.66 0.08
0.15 0.30 0.15 -2.13
-0.02 1.03 -1.43 0.50 ;End of matrix A

4 6 :Values of M and N, Example 2
-5.42 3.28 -3.68 0.27 2.06 0.46
-1.65 3.40 -3.20 -1.03 -4.06 -0.01
-0.37 2.35 1.90 4.31 -1.76 1.13
-3.15 -0.11 1.99 -2.70 0.26 4.50 ;End of matrix A

9.3 Program Results
f08kgc Example Program Results

Example 1: bidiagonal matrix B
Diagonal
3.6177 -2.4161 1.9213 -1.4265
Super-diagonal
1.2587 -1.5262 1.1895

Example 1: matrix Q

Example 2: bidiagonal matrix B
Diagonal
-7.7724  6.1573  -6.0576  5.7933
Super-diagonal
1.1926  0.5734  -1.9143

Example 2: matrix $P^T$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.7104</td>
<td>0.4299</td>
<td>-0.4824</td>
<td>0.0354</td>
<td>0.2700</td>
<td>0.0603</td>
</tr>
<tr>
<td>2</td>
<td>0.3583</td>
<td>0.1382</td>
<td>-0.4110</td>
<td>0.4044</td>
<td>0.0951</td>
<td>-0.7148</td>
</tr>
<tr>
<td>3</td>
<td>-0.0507</td>
<td>0.4244</td>
<td>0.3795</td>
<td>0.7402</td>
<td>-0.2773</td>
<td>0.2203</td>
</tr>
<tr>
<td>4</td>
<td>0.2442</td>
<td>0.4016</td>
<td>0.4158</td>
<td>-0.1354</td>
<td>0.7666</td>
<td>-0.0137</td>
</tr>
</tbody>
</table>