NAG Library Function Document

nag_dormqr (f08agc)

1 Purpose
nag_dormqr (f08agc) multiplies an arbitrary real matrix \( C \) by the real orthogonal matrix \( Q \) from a \( QR \) factorization computed by nag_dgeqrf (f08aec), nag_dgeqpf (f08bec) or nag_dgeqp3 (f08bfc).

2 Specification

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

void nag_dormqr (Nag_OrderType order, 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_dormqr (f08agc) is intended to be used after a call to nag_dgeqrf (f08aec), nag_dgeqpf (f08bec) or nag_dgeqp3 (f08bfc) which perform a \( QR \) factorization of a real matrix \( A \). The orthogonal matrix \( Q \) is represented as a product of elementary reflectors.

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

\[
QC, Q^T C, CQ \quad \text{or} \quad CQ^T,
\]

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

A common application of this function is in solving linear least squares problems, as described in the f08 Chapter Introduction and illustrated in Section 10 in nag_dgeqrf (f08aec).

4 References

5 Arguments

1: \( \text{order} \) – Nag_OrderType

On entry: the \( \text{order} \) 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 \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

2: \( \text{side} \) – Nag_SideType

On entry: indicates how \( Q \) or \( Q^T \) is to be applied to \( C \).

\( \text{side} = \text{Nag_LeftSide} \)

\( Q \) or \( Q^T \) is applied to \( C \) from the left.

\( \text{side} = \text{Nag_RightSide} \)

\( Q \) or \( Q^T \) is applied to \( C \) from the right.

Constraint: \( \text{side} = \text{Nag_LeftSide} \) or \( \text{Nag_RightSide} \).
3: trans – Nag_TransType
   On entry: indicates whether $Q$ or $Q^T$ is to be applied to $C$.
   
   trans = Nag_NoTrans
   $Q$ is applied to $C$.
   
   trans = Nag_Trans
   $Q^T$ is applied to $C$.
   
   Constraint: trans = Nag_NoTrans or Nag_Trans.

4: m – Integer
   On entry: $m$, the number of rows of the matrix $C$.
   
   Constraint: $m \geq 0$.

5: n – Integer
   On entry: $n$, the number of columns of the matrix $C$.
   
   Constraint: $n \geq 0$.

6: k – Integer
   On entry: $k$, the number of elementary reflectors whose product defines the matrix $Q$.
   
   Constraints:
   
   if side = Nag_LeftSide, $m \geq k \geq 0$;
   if side = Nag_RightSide, $n \geq k \geq 0$.

7: a[dim] – const double
   
   Note: the dimension, dim, of the array a must be at least
   
   $\max(1, pda \times k)$ when order = Nag_ColMajor;
   $\max(1, m \times pda)$ when order = Nag_RowMajor and side = Nag_LeftSide;
   $\max(1, n \times pda)$ when order = Nag_RowMajor and side = Nag_RightSide.
   
   On entry: details of the vectors which define the elementary reflectors, as returned by nag_dgeqrf (f08aec), nag_dgeqpf (f08bec) or nag_dgeqp3 (f08bfc).

8: pda – Integer
   On entry: the stride separating row or column elements (depending on the value of order) in the array a.
   
   Constraints:
   
   if order = Nag_ColMajor,
   
   if side = Nag_LeftSide, $pda \geq \max(1, m)$;
   if side = Nag_RightSide, $pda \geq \max(1, n)$;
   if order = Nag_RowMajor, $pda \geq \max(1, k)$.

9: tau[dim] – const double
   
   Note: the dimension, dim, of the array tau must be at least $\max(1, k)$.
   
   On entry: further details of the elementary reflectors, as returned by nag_dgeqrf (f08aec), nag_dgeqpf (f08bec) or nag_dgeqp3 (f08bfc).
10: \(c[\text{dim}]\) – double

Input/Output

**Note:** the dimension, \(\text{dim}\), of the array \(c\) must be at least
\[
\max(1, \text{pdc} \times n) \text{ when } \text{order} = \text{Nag\_ColMajor};
\]
\[
\max(1, m \times \text{pdc}) \text{ when } \text{order} = \text{Nag\_RowMajor}.
\]
The \((i,j)\)th element of the matrix \(C\) is stored in
\[
c[(j - 1) \times \text{pdc} + i - 1] \text{ when } \text{order} = \text{Nag\_ColMajor};
\]
\[
c[(i - 1) \times \text{pdc} + j - 1] \text{ when } \text{order} = \text{Nag\_RowMajor}.
\]

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

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

11: \(\text{pdc}\) – Integer

Input

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

**Constraints:**
\[
\begin{align*}
\text{if } \text{order} = \text{Nag\_ColMajor}, & \quad \text{pdc} \geq \max(1, m); \\
\text{if } \text{order} = \text{Nag\_RowMajor}, & \quad \text{pdc} \geq \max(1, n).
\end{align*}
\]

12: \(\text{fail}\) – 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.

See Section 3.2.1.2 in the Essential Introduction for further information.

**NE\_BAD\_PARAM**

On entry, argument \(<\text{value}\>\) had an illegal value.

**NE\_ENUM\_INT\_3**

On entry, \(\text{side} = <\text{value}\>, \ m = <\text{value}\>, \ n = <\text{value}\>\) and \(k = <\text{value}\>\).

Constraint: if \(\text{side} = \text{Nag\_LeftSide}, \ m \geq k \geq 0;\)
if \(\text{side} = \text{Nag\_RightSide}, \ n \geq k \geq 0.\)

On entry, \(\text{side} = <\text{value}\>, \ m = <\text{value}\>, \ n = <\text{value}\>\) and \(\text{pda} = <\text{value}\>\).

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\_INT**

On entry, \(m = <\text{value}\>.\)

Constraint: \(m \geq 0.\)

On entry, \(n = <\text{value}\>.\)

Constraint: \(n \geq 0.\)

On entry, \(\text{pda} = <\text{value}\>.\)

Constraint: \(\text{pda} > 0.\)

On entry, \(\text{pdc} = <\text{value}\>.\)

Constraint: \(\text{pdc} > 0.\)
On entry, \( pda = \langle \text{value} \rangle \) and \( k = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, k) \).

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

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

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.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

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.

nag_dormqr (f08agc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dormqr (f08agc) 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 X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

The total number of floating-point operations is approximately \( 2nk(2m - k) \) if \( \text{side} = \text{Nag LeftSide} \) and \( 2nk(2n - k) \) if \( \text{side} = \text{Nag RightSide} \). The complex analogue of this function is nag_zunmqr (f08auc).

See Section 10 in nag_dgeqrf (f08aec).