

NAG C 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) or nag_dgeqpf (f08bec).

2 Specification

```
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) or nag_dgeqpf (f08bec), 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

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

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 9 of the document for nag_dgeqrf (f08aec).

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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 **trans** = **Nag_NoTrans**, Q is applied to C ;
 if **trans** = **Nag_Trans**, Q^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 .

Constraint: **m** ≥ 0 .

5: **n** – Integer *Input*

On entry: n , the number of columns of the matrix C .

Constraint: **n** ≥ 0 .

6: **k** – Integer *Input*

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*] – double *Input/Output*

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

$\max(1, \mathbf{pda} \times \mathbf{k})$ when **order** = **Nag_ColMajor**;
 $\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = **Nag_RowMajor** and **side** = **Nag_LeftSide**;
 $\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = **Nag_RowMajor** and **side** = **Nag_RightSide**.

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_dgeqr** (f08aec) or **nag_dgeqpf** (f08bec).

On exit: used as internal workspace prior to being restored and hence is unchanged.

8: **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 **side** = **Nag_LeftSide**, **pda** $\geq \max(1, \mathbf{m})$;
 if **side** = **Nag_RightSide**, **pda** $\geq \max(1, \mathbf{n})$;
 if **order** = **Nag_RowMajor**, **pda** $\geq \max(1, \mathbf{k})$.

9: **tau**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **tau** must be at least $\max(1, \mathbf{k})$.

On entry: further details of the elementary reflectors, as returned by **nag_dgeqr** (f08aec) or **nag_dgeqpf** (f08bec).

10: **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** and at least $\max(1, \mathbf{pdc} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

If **order** = **Nag_ColMajor**, the (i, j) th element of the matrix C is stored in **c**[(*j* – 1) \times **pdc** + *i* – 1] and if **order** = **Nag_RowMajor**, the (i, j) th element of the matrix C is stored in **c**[(*i* – 1) \times **pdc** + *j* – 1].

On entry: the m by n matrix C .

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

11: **pdc** – Integer *Input*

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

Constraints:

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

12: **fail** – **NagError** * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

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, **pda** = $\langle value \rangle$.

Constraint: **pda** > 0 .

On entry, **pdc** = $\langle value \rangle$.

Constraint: **pdc** > 0 .

NE_INT_2

On entry, **pda** = $\langle value \rangle$, $\mathbf{k} = \langle value \rangle$.

Constraint: **pda** $\geq \max(1, k)$.

On entry, **pdc** = $\langle value \rangle$, $\mathbf{m} = \langle value \rangle$.

Constraint: **pdc** $\geq \max(1, m)$.

On entry, **pdc** = $\langle value \rangle$, $\mathbf{n} = \langle value \rangle$.

Constraint: **pdc** $\geq \max(1, n)$.

NE_ENUM_INT_3

On entry, **side** = $\langle value \rangle$, $\mathbf{m} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$, $\mathbf{k} = \langle value \rangle$.

Constraint: if **side** = **Nag_LeftSide**, $\mathbf{m} \geq \mathbf{k} \geq 0$;

if **side** = **Nag_RightSide**, $\mathbf{n} \geq \mathbf{k} \geq 0$.

On entry, **side** = $\langle value \rangle$, $\mathbf{m} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **side** = **Nag_LeftSide**, **pda** $\geq \max(1, m)$;

if **side** = **Nag_RightSide**, **pda** $\geq \max(1, n)$.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 ϵ is the *machine precision*.

8 Further Comments

The total number of floating-point operations is approximately $2nk(2m - k)$ if `side = Nag_LeftSide` and $2mk(2n - k)$ if `side = Nag_RightSide`.

The complex analogue of this function is `nag_zunmqr` (f08auc).

9 Example

See Section 9 of the document for `nag_dgeqrf` (f08aec).
