

# NAG Library Function Document

## nag\_dormlq (f08akc)

### 1 Purpose

nag\_dormlq (f08akc) multiplies an arbitrary real matrix  $C$  by the real orthogonal matrix  $Q$  from an  $LQ$  factorization computed by nag\_dgelqf (f08ahc).

### 2 Specification

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

void nag_dormlq (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\_dormlq (f08akc) is intended to be used after a call to nag\_dgelqf (f08ahc), which performs an  $LQ$  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).

### 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  $Q^T$  is to be applied to  $C$ .

**side** = Nag\_LeftSide

$Q$  or  $Q^T$  is applied to  $C$  from the left.

**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  $C$ .  
**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 *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $C$ .  
*Constraint:* **m**  $\geq 0$ .
- 5: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $C$ .  
*Constraint:* **n**  $\geq 0$ .
- 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*] – 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 and **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{k} \times \mathbf{pda})$  when **side** = Nag\_LeftSide and **order** = Nag\_RowMajor;  
 $\max(1, \mathbf{pda} \times \mathbf{n})$  when **side** = Nag\_RightSide and **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{k} \times \mathbf{pda})$  when **side** = Nag\_RightSide and **order** = Nag\_RowMajor.  
*On entry:* details of the vectors which define the elementary reflectors, as returned by nag\_dgelqf (f08ahc).
- 8: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **a**.  
*Constraints:*  
if **order** = Nag\_ColMajor, **pda**  $\geq$   $\max(1, \mathbf{k})$ ;  
if **order** = Nag\_RowMajor,  
if **side** = Nag\_LeftSide, **pda**  $\geq$   $\max(1, \mathbf{m})$ ;  
if **side** = Nag\_RightSide, **pda**  $\geq$   $\max(1, \mathbf{n})$ .
- 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\_dgelqf (f08ahc).

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;  
 $\max(1, \mathbf{m} \times \mathbf{pdc})$  when **order** = Nag\_RowMajor.

The (*i*, *j*)th element of the matrix *C* is stored in

**c**[(*j* – 1) × **pdc** + *i* – 1] when **order** = Nag\_ColMajor;  
**c**[(*i* – 1) × **pdc** + *j* – 1] when **order** = 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 **side** and **trans**.

11: **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, **pdc** ≥ max(1, **m**);  
 if **order** = Nag\_RowMajor, **pdc** ≥ max(1, **n**).

12: **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 **k** = *⟨value⟩*.

Constraint: if **side** = Nag\_LeftSide, **m** ≥ **k** ≥ 0;

if **side** = Nag\_RightSide, **n** ≥ **k** ≥ 0.

On entry, **side** = *⟨value⟩*, **pda** = *⟨value⟩*, **m** = *⟨value⟩* and **n** = *⟨value⟩*.

Constraint: if **side** = Nag\_LeftSide, **pda** ≥ max(1, **m**);

if **side** = Nag\_RightSide, **pda** ≥ max(1, **n**).

### NE\_INT

On entry, **m** = *⟨value⟩*.

Constraint: **m** ≥ 0.

On entry, **n** = *⟨value⟩*.

Constraint: **n** ≥ 0.

On entry, **pda** = *⟨value⟩*.

Constraint: **pda** > 0.

On entry, **pdc** = *⟨value⟩*.

Constraint: **pdc** > 0.

**NE\_INT\_2**

On entry, **pda** =  $\langle value \rangle$  and **k** =  $\langle value \rangle$ .  
 Constraint: **pda**  $\geq \max(1, \mathbf{k})$ .

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

On entry, **pdc** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **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\_dormlq (f08akc) is not threaded by NAG in any implementation.

nag\_dormlq (f08akc) 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  $2nk(2m - k)$  if **side** = Nag\_LeftSide and  $2mk(2n - k)$  if **side** = Nag\_RightSide.

The complex analogue of this function is nag\_zunmlq (f08axc).

**10 Example**

See Section 10 in nag\_dgelqf (f08ahc).

---