# NAG Library Function Document

# nag_zunmlq (f08axc)

## 1 Purpose

nag_zunmlq (f08axc) multiplies an arbitrary complex matrix $C$ by the complex unitary matrix $Q$ from an $LQ$ factorization computed by nag_zgelqf (f08avc).

## 2 Specification

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

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

## 3 Description

nag_zunmlq (f08axc) is intended to be used after a call to nag_zgelqf (f08avc), which performs an $LQ$ factorization of a complex matrix $A$. The unitary matrix $Q$ is represented as a product of elementary reflectors.

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

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

overwriting the result on $C$ (which may be any complex 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 $Q^{\mathrm{H}}$ is to be applied to $C$.

**side** = Nag_LeftSide
: $Q$ or $Q^{\mathrm{H}}$ is applied to $C$ from the left.

**side** = Nag_RightSide
: $Q$ or $Q^{\mathrm{H}}$ 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 $Q^{\mathrm{H}}$ is to be applied to $C$.

**trans** = Nag_NoTrans
$Q$ is applied to $C$.

**trans** = Nag_ConjTrans
$Q^{\mathrm{H}}$ is applied to $C$.

*Constraint*: **trans** = Nag_NoTrans or Nag_ConjTrans.

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 Complex *Input*

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

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

*On entry*: details of the vectors which define the elementary reflectors, as returned by nag_zgelqf (f08avc).

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** $\geq \max(1, \textbf{k})$;
if **order** = Nag_RowMajor,

if **side** = Nag_LeftSide, **pda** $\geq \max(1, \textbf{m})$;
if **side** = Nag_RightSide, **pda** $\geq \max(1, \textbf{n})$..

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

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

*On entry*: further details of the elementary reflectors, as returned by nag_zgelqf (f08avc).

10: **c**[$dim$] – Complex *Input/Output*

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

max(1, **pdc** × **n**) when **order** = Nag_ColMajor;
max(1, **m** × **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 $Q^{H}C$ or $CQ$ or $CQ^{H}$ 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_zunmlq (f08axc) is not threaded by NAG in any implementation.

nag_zunmlq (f08axc) 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 real floating-point operations is approximately $8nk(2m-k)$ if **side** = Nag_LeftSide and $8mk(2n-k)$ if **side** = Nag_RightSide.

The real analogue of this function is nag_dormlq (f08akc).

## 10 Example

See Section 10 in nag_zgelqf (f08avc).