NAG Library Function Document

nag_zunmql (f08cuc)

1 Purpose

nag_zunmql (f08cuc) multiplies a general complex m by n matrix C by the complex unitary matrix Q from a QL factorization computed by nag_zgeqlf (f08csc).

2 Specification

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

```
void nag_zunmql (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_zunmql (f08cuc) is intended to be used following a call to nag_zgeqlf (f08csc), which performs a QL factorization of a complex matrix A and represents the unitary matrix Q as a product of elementary reflectors.

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

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

overwriting the result on C, which may be any complex rectangular m by n 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_zgeqlf (f08csc).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

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^{H} is to be applied to C.

side = Nag_LeftSide
Q or Q^{H} is applied to C from the left.

side = Nag_RightSide
Q or Q^{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^{H} is to be applied to C.

trans = Nag_NoTrans
Q is applied to C.

trans = Nag_ConjTrans
Q^{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: $\mathbf{m} \geq 0$.

5: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

6: **k** – Integer *Input*

On entry: k, the number of elementary reflectors whose product defines the matrix Q.

Constraints:

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

7: **a**[*dim*] – const Complex *Input*

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

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

On entry: details of the vectors which define the elementary reflectors, as returned by nag_zgeqlf (f08csc).

On exit: is modified by nag_zunmql (f08cuc) but restored on exit.

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,

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

if **order** = Nag_RowMajor, $\mathbf{pda} \geq \max(1, \mathbf{k})$.

9: **tau**[*dim*] – const Complex *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_zgeqlf (f08csc).

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.
See Section 3.2.1.2 in the Essential Introduction for further information.

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*⟩, **m** = ⟨*value*⟩, **pda** = ⟨*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** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: **pda** $\geq \max(1, \mathbf{k})$.

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

On entry, **pdc** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
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.

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

NE_NO_LICENCE

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.

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 Parallelism and Performance

nag_zunmql (f08cuc) is not threaded by NAG in any implementation.

nag_zunmql (f08cuc) 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.

9 Further Comments

The total number of 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_dormql (f08cgc).

10 Example

See Section 10 in nag_zgeqlf (f08csc).