NAG Library Function Document

nag_zunmrq (f08cxc)

1 Purpose

nag_zunmrq (f08cxc) multiplies a general complex \( m \) by \( n \) matrix \( C \) by the complex unitary matrix \( Q \) from an \( RQ \) factorization computed by nag_zgerqf (f08cvc).

2 Specification

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

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

3 Description

nag_zunmrq (f08cxc) is intended to be used following a call to nag_zgerqf (f08cvc), which performs an \( RQ \) 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

\[
QC, \quad Q^HC, \quad CQ, \quad CQ^H
\]

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

A common application of this function is in solving underdetermined linear least squares problems, as described in the f08 Chapter Introduction, and illustrated in Section 10 in nag_zgerqf (f08cvc).

4 References


5 Arguments

1: \textbf{order} \quad \textbf{Nag_OrderType} \quad \textit{Input}

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

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{side} \quad \textbf{Nag_SideType} \quad \textit{Input}

\textit{On entry:} indicates how \( Q \) or \( Q^H \) is to be applied to \( C \).

\textbf{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.

Constrain: 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 $C$.

Constrain: 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 $Q$.
    Constrains:
    if side = Nag_LeftSide, $m \geq k \geq 0$;
    if side = Nag_RightSide, $n \geq k \geq 0$.

7: a[dim] – Complex
    Input/Output
    Note: the dimension, dim, of the array a must be at least
    max(1, pda \times m) when side = Nag_LeftSide and order = Nag_ColMajor;
    max(1, k \times pda) when side = Nag_LeftSide and order = Nag_RowMajor;
    max(1, pda \times n) when side = Nag_RightSide and order = Nag_ColMajor;
    max(1, k \times pda) when side = Nag_RightSide and order = Nag_RowMajor.

The $(i, j)$th element of the matrix $A$ is stored in
    a[j - 1] \times pda + i - 1] when order = Nag_ColMajor;
    a[i - 1] \times pda + j - 1] when order = Nag_RowMajor.

On entry: the $i$th row of $a$ must contain the vector which defines the elementary reflector $H_i$, for
$i = 1, 2, \ldots, k$, as returned by nag_zgerqf (f08cvc).

On exit: is modified by nag_zunmrq (f08cxc) 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.
    Constrains: 
    if order = Nag_ColMajor, pda \geq max(1, k);
    if order = Nag_RowMajor,
        if side = Nag_LeftSide, pda \geq max(1, m);
        if side = Nag_RightSide, pda \geq max(1, n).
9: \( \text{tau}[\text{dim}] \) – const Complex \hspace{1cm} \text{Input}

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

On entry: \( \text{tau}[i-1] \) must contain the scalar factor of the elementary reflector \( H_i \), as returned by \( \text{nag_zgerqf} \) (f08cvc).

10: \( \text{c}[\text{dim}] \) – Complex \hspace{1cm} \text{Input/Output}

Note: the dimension, \( \text{dim} \), of the array \( \text{c} \) must be at least:

\[
\max(1, p\text{d}c \times n) \quad \text{when} \quad \text{order} = \text{Nag} \text{ColMajor};
\]
\[
\max(1, m \times p\text{d}c) \quad \text{when} \quad \text{order} = \text{Nag} \text{RowMajor}.
\]

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

\[
\text{c}[(j-1) \times p\text{d}c + i-1] \quad \text{when} \quad \text{order} = \text{Nag} \text{ColMajor};
\]
\[
\text{c}[(i-1) \times p\text{d}c + j-1] \quad \text{when} \quad \text{order} = \text{Nag} \text{RowMajor}.
\]

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

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

11: \( \text{pdc} \) – Integer \hspace{1cm} \text{Input}

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

Constraints:

\[
\begin{align*}
\text{if} \quad \text{order} = \text{Nag} \text{ColMajor}, & \quad p\text{d}c \geq \max(1, m); \\
\text{if} \quad \text{order} = \text{Nag} \text{RowMajor}, & \quad p\text{d}c \geq \max(1, n).
\end{align*}
\]

12: \( \text{fail} \) – NagError * \hspace{1cm} \text{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 \( \langle \text{value} \rangle \) had an illegal value.

**NE_ENUM_INT_3**

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

Constraint: if \( \text{side} = \text{Nag} \text{LeftSide} \), \( m \geq k \geq 0 \);

if \( \text{side} = \text{Nag} \text{RightSide} \), \( n \geq k \geq 0 \).

On entry, \( \text{side} = \langle \text{value} \rangle, p\text{d}a = \langle \text{value} \rangle, m = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: if \( \text{side} = \text{Nag} \text{LeftSide} \), \( p\text{d}a \geq \max(1, m) \);

if \( \text{side} = \text{Nag} \text{RightSide} \), \( p\text{d}a \geq \max(1, n) \).

**NE_INT**

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).
On entry, $pda = \langle value \rangle$.
Constraint: $pda > 0$.

On entry, $pdc = \langle value \rangle$.
Constraint: $pdc > 0$.

**NE_INT_2**

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

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

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

**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 $\epsilon$ is the *machine precision*.

8 Parallelism and Performance

*nag_zunmrq* (f08cxc) is not threaded by NAG in any implementation.

*nag_zunmrq* (f08cxc) 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 $\text{side} = \text{Nag_LeftSide}$ and $8mk(2n - k)$ if $\text{side} = \text{Nag_RightSide}$.

The real analogue of this function is *nag_dormrq* (f08ckc).

10 Example

See Section 10 in *nag_zgerqf* (f08cvc).