# NAG Library Function Document

# nag_dgemqrt (f08acc)

## 1 Purpose

nag_dgemqrt (f08acc) multiplies an arbitrary real matrix $C$ by the real orthogonal matrix $Q$ from a $QR$ factorization computed by nag_dgeqrt (f08abc).

## 2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dgemqrt (Nag_OrderType order, Nag_SideType side,
     Nag_TransType trans, Integer m, Integer n, Integer k, Integer nb,
     const double v[], Integer pdv, const double t[], Integer pdt,
     double c[], Integer pdc, NagError *fail)
```

## 3 Description

nag_dgemqrt (f08acc) is intended to be used after a call to nag_dgeqrt (f08abc) which performs 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

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

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 10 in nag_dgeqrt (f08abc).

## 4 References

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th 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 $Q^{\mathrm{T}}$ is to be applied to $C$.

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

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

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

**trans** = Nag_Trans
$Q^{\mathrm{T}}$ 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*: $\mathbf{m} \geq 0$.

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 $Q$. Usually $\mathbf{k} = \min(m_A, n_A)$ where $m_A$, $n_A$ are the dimensions of the matrix $A$ supplied in a previous call to nag_dgeqrt (f08abc).

*Constraints*:

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

7: **nb** – Integer *Input*

*On entry*: the block size used in the $QR$ factorization performed in a previous call to nag_dgeqrt (f08abc); this value must remain unchanged from that call.

*Constraints*:

$\mathbf{nb} \geq 1$;
if $\mathbf{k} > 0$, $\mathbf{nb} \leq \mathbf{k}$.

8: **v**[$dim$] – const double *Input*

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

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

*On entry*: details of the vectors which define the elementary reflectors, as returned by nag_dgeqrt (f08abc) in the first $k$ columns of its array argument **a**.

9: **pdv** – Integer *Input*

*On entry*: the stride separating row or column elements (depending on the value of **order**) in the array **v**.

*Constraints*:

if **order** = Nag_ColMajor,

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

10: **t**[*dim*] – const double *Input*

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

max(1, **pdt** × **k**) when **order** = Nag_ColMajor;
max(1, **nb** × **pdt**) when **order** = Nag_RowMajor.

The $(i, j)$th element of the matrix $T$ is stored in

**t**[$(j-1) \times$ **pdt** $+ i - 1$] when **order** = Nag_ColMajor;
**t**[$(i-1) \times$ **pdt** $+ j - 1$] when **order** = Nag_RowMajor.

*On entry*: further details of the orthogonal matrix $Q$ as returned by nag_dgeqrt (f08abc). The number of blocks is $b = \left\lceil \frac{k}{\mathbf{nb}} \right\rceil$, where $k = \min(m, n)$ and each block is of order **nb** except for the last block, which is of order $k - (b-1) \times$ **nb**. For the $b$ blocks the upper triangular block reflector factors $\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_b$ are stored in the **nb** by $n$ matrix $T$ as $\boldsymbol{T} = [\boldsymbol{T}_1 | \boldsymbol{T}_2 | \ldots | \boldsymbol{T}_b]$.

11: **pdt** – Integer *Input*

*On entry*: the stride separating row or column elements (depending on the value of **order**) in the array **t**.

*Constraints*:

if **order** = Nag_ColMajor, **pdt** ≥ **nb**;
if **order** = Nag_RowMajor, **pdt** ≥ max(1, **k**).

12: **c**[*dim*] – double *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**[$(j-1) \times$ **pdc** $+ i - 1$] when **order** = Nag_ColMajor;
**c**[$(i-1) \times$ **pdc** $+ j - 1$] when **order** = Nag_RowMajor.

*On entry*: the $m$ by $n$ matrix $C$.

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

13: **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**).

14: **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** $= \langle value \rangle$, **m** $= \langle value \rangle$, **n** $= \langle value \rangle$ and **k** $= \langle value \rangle$.
Constraint: if **side** = Nag_LeftSide, **m** $\geq$ **k** $\geq 0$;
if **side** = Nag_RightSide, **n** $\geq$ **k** $\geq 0$.

On entry, **side** $= \langle value \rangle$, **m** $= \langle value \rangle$, **n** $= \langle value \rangle$ and **pdv** $= \langle value \rangle$.
Constraint: if **side** = Nag_LeftSide, **pdv** $\geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, **pdv** $\geq \max(1, \mathbf{n})$.

**NE_INT**

On entry, **m** $= \langle value \rangle$.
Constraint: **m** $\geq 0$.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** $\geq 0$.

**NE_INT_2**

On entry, **nb** $= \langle value \rangle$ and **k** $= \langle value \rangle$.
Constraint: **nb** $\geq 1$ and
if **k** $> 0$, **nb** $\leq$ **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** $\geq \max(1, \mathbf{n})$.

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

On entry, **pdt** $= \langle value \rangle$ and **nb** $= \langle value \rangle$.
Constraint: **pdt** $\geq$ **nb**.

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

**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_dgemqrt (f08acc) is not threaded by NAG in any implementation.

nag_dgemqrt (f08acc) 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 $2nk(2m-k)$ if **side** = Nag_LeftSide and $2mk(2n-k)$ if **side** = Nag_RightSide.

The complex analogue of this function is nag_zgemqrt (f08aqc).

## 10 Example

See Section 10 in nag_dgeqrt (f08abc).