NAG Library Function Document

nag_dtpmqrt (f08bcc)

1 Purpose

nag_dtpmqrt (f08bcc) multiplies an arbitrary real matrix C by the real orthogonal matrix Q from a QR factorization computed by nag_dtpqrt (f08bbc).

2 Specification

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

void nag_dtpmqrt (Nag_OrderType order, Nag_SideType side,
     Nag_TransType trans, Integer m, Integer n, Integer k, Integer l,
     Integer nb, const double v[], Integer pdv, const double t[],
     Integer pdt, double c1[], Integer pdc1, double c2[], Integer pdc2,
     NagError *fail)
```

3 Description

nag_dtpmqrt (f08bcc) is intended to be used after a call to nag_dtpqrt (f08bbc) which performs a QR factorization of a triangular-pentagonal matrix containing an upper triangular matrix A over a pentagonal matrix B. The orthogonal matrix Q is represented as a product of elementary reflectors.

This function may be used to form the matrix products

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

where the real rectangular m_c by n_c matrix C is split into component matrices C_1 and C_2.

If Q is being applied from the left (QC or $Q^{\mathrm{T}}C$) then

$$C = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

where C_1 is k by n_c, C_2 is m_v by n_c, $m_c = k + m_v$ is fixed and m_v is the number of rows of the matrix V containing the elementary reflectors (i.e., **m** as passed to nag_dtpqrt (f08bbc)); the number of columns of V is n_v (i.e., **n** as passed to nag_dtpqrt (f08bbc)).

If Q is being applied from the right (CQ or CQ^{T}) then

$$C = \begin{pmatrix} C_1 & C_2 \end{pmatrix}$$

where C_1 is m_c by k, and C_2 is m_c by m_v and $n_c = k + m_v$ is fixed.

The matrices C_1 and C_2 are overwriten by the result of the matrix product.

A common application of this routine is in updating the solution of a linear least squares problem as illustrated in Section 10 in nag_dtpqrt (f08bbc).

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^{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: the number of rows of the matrix C_2, that is,

if **side** = Nag_LeftSide
then m_v, the number of rows of the matrix V;

if **side** = Nag_RightSide
then m_c, the number of rows of the matrix C.

Constraint: **m** ≥ 0.

5: **n** – Integer *Input*

On entry: the number of columns of the matrix C_2, that is,

if **side** = Nag_LeftSide
then n_c, the number of columns of the matrix C;

if **side** = Nag_RightSide
then n_v, the number of columns of the matrix V.

Constraint: **n** ≥ 0.

6: **k** – Integer *Input*

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

Constraint: **k** ≥ 0.

7: **l** – Integer *Input*

On entry: l, the number of rows of the upper trapezoidal part of the pentagonal composite matrix V, passed (as **b**) in a previous call to nag_dtpqrt (f08bbc). This must be the same value used in the previous call to nag_dtpqrt (f08bbc) (see **l** in nag_dtpqrt (f08bbc)).

Constraint: $0 \leq \mathbf{l} \leq \mathbf{k}$.

8: **nb** – Integer *Input*

On entry: nb, the blocking factor used in a previous call to nag_dtpqrt (f08bbc) to compute the QR factorization of a triangular-pentagonal matrix containing composite matrices A and B.

Constraints:

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

9: **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.

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

$\mathbf{v}[(j-1) \times \mathbf{pdv} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{v}[(i-1) \times \mathbf{pdv} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the m_v by n_v matrix V; this should remain unchanged from the array **b** returned by a previous call to nag_dtpqrt (f08bbc).

10: **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, $\mathbf{pdv} \geq \max(1, \mathbf{k})$.

11: **t**[dim] – const double *Input*

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

$\max(1, \mathbf{pdt} \times \mathbf{k})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{nb} \times \mathbf{pdt})$ when **order** = Nag_RowMajor.

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

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

On entry: this must remain unchanged from a previous call to nag_dtpqrt (f08bbc) (see **t** in nag_dtpqrt (f08bbc)).

12: **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** $\geq$ **nb**;
if **order** $=$ Nag_RowMajor, **pdt** $\geq \max(1, \mathbf{k})$.

13: **c1**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **c1** must be at least

$\max(1, \mathbf{pdc1} \times \mathbf{n})$ when **side** $=$ Nag_LeftSide and **order** $=$ Nag_ColMajor;
$\max(1, \mathbf{k} \times \mathbf{pdc1})$ when **side** $=$ Nag_LeftSide and **order** $=$ Nag_RowMajor;
$\max(1, \mathbf{pdc1} \times \mathbf{k})$ when **side** $=$ Nag_RightSide and **order** $=$ Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdc1})$ when **side** $=$ Nag_RightSide and **order** $=$ Nag_RowMajor.

On entry: C_1, the first part of the composite matrix C.

if **side** $=$ Nag_LeftSide
then **c1** contains the first k rows of C;

if **side** $=$ Nag_RightSide
then **c1** contains the first k columns of C.

On exit: **c1** is overwritten by the corresponding block of QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T}.

14: **pdc1** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c1**.

Constraints:

if **order** $=$ Nag_ColMajor,

if **side** $=$ Nag_LeftSide, **pdc1** $\geq \max(1, \mathbf{k})$;
if **side** $=$ Nag_RightSide, **pdc1** $\geq \max(1, \mathbf{m})$.;

if **order** $=$ Nag_RowMajor,

if **side** $=$ Nag_LeftSide, **pdc1** $\geq \max(1, \mathbf{n})$;
if **side** $=$ Nag_RightSide, **pdc1** $\geq \max(1, \mathbf{k})$..

15: **c2**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **c2** must be at least

$\max(1, \mathbf{pdc2} \times \mathbf{n})$ when **order** $=$ Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pdc2})$ when **order** $=$ Nag_RowMajor.

On entry: C_2, the second part of the composite matrix C.

if **side** $=$ Nag_LeftSide
then **c2** contains the remaining m_v rows of C;

if **side** $=$ Nag_RightSide
then **c2** contains the remaining m_v columns of C;

On exit: **c2** is overwritten by the corresponding block of QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T}.

16: **pdc2** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c2**.

Constraints:

if **order** $=$ Nag_ColMajor, **pdc2** $\geq \max(1, \mathbf{m})$;
if **order** $=$ Nag_RowMajor, **pdc2** $\geq \max(1, \mathbf{n})$.

17: **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*⟩, **k** = ⟨*value*⟩, **m** = ⟨*value*⟩ and **pdc1** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, **pdc1** $\geq \max(1, \mathbf{k})$;
if **side** = Nag_RightSide, **pdc1** $\geq \max(1, \mathbf{m})$.

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

On entry, **side** = ⟨*value*⟩, **pdc1** = ⟨*value*⟩, **n** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, **pdc1** $\geq \max(1, \mathbf{n})$;
if **side** = Nag_RightSide, **pdc1** $\geq \max(1, \mathbf{k})$.

NE_INT

On entry, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **l** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: $0 \leq \mathbf{l} \leq \mathbf{k}$.

On entry, **m** = ⟨*value*⟩ and **pdc2** = ⟨*value*⟩.
Constraint: **pdc2** $\geq \max(1, \mathbf{m})$.

On entry, **nb** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: **nb** ≥ 1 and
if $\mathbf{k} > 0$, $\mathbf{nb} \leq \mathbf{k}$.

On entry, **pdc2** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdc2** $\geq \max(1, \mathbf{n})$.

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

On entry, **pdt** = ⟨*value*⟩ and **nb** = ⟨*value*⟩.
Constraint: **pdt** $\geq$ **nb**.

On entry, **pdv** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
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 ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_dtpmqrt (f08bcc) is not threaded by NAG in any implementation.

nag_dtpmqrt (f08bcc) 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_ztpmqrt (f08bqc).

10 Example

See Section 10 in nag_dtpqrt (f08bbc).
