NAG Toolbox

nag_lapack_dtpmqrt (f08bc)

1 Purpose

nag_lapack_dtpmqrt (f08bc) multiplies an arbitrary real matrix C by the real orthogonal matrix Q from a QR factorization computed by nag_lapack_dtpqrt (f08bb).

2 Syntax

```
[c1, c2, info] = nag_lapack_dtpmqrt(side, trans, l, v, t, c1, c2, 'm', m, 'n', n, 'k', k, 'nb', nb)
[c1, c2, info] = f08bc(side, trans, l, v, t, c1, c2, 'm', m, 'n', n, 'k', k, 'nb', nb)
```

3 Description

nag_lapack_dtpmqrt (f08bc) is intended to be used after a call to nag_lapack_dtpqrt (f08bb) 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_lapack_dtpqrt (f08bb)); the number of columns of V is n_v (i.e., **n** as passed to nag_lapack_dtpqrt (f08bb)).

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_lapack_dtpqrt (f08bb).

4 References

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th Edition) Johns Hopkins University Press, Baltimore

5 Parameters

5.1 Compulsory Input Parameters

1: **side** – CHARACTER(1)

Indicates how Q or Q^{T} is to be applied to C.

side = 'L'
Q or Q^{T} is applied to C from the left.

side = 'R'
Q or Q^{T} is applied to C from the right.

Constraint: **side** = 'L' or 'R'.

2: **trans** – CHARACTER(1)

Indicates whether Q or Q^{T} is to be applied to C.

trans = 'N'
Q is applied to C.

trans = 'T'
Q^{T} is applied to C.

Constraint: **trans** = 'N' or 'T'.

3: **l** – INTEGER

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_lapack_dtpqrt (f08bb). This must be the same value used in the previous call to nag_lapack_dtpqrt (f08bb) (see **l** in nag_lapack_dtpqrt (f08bb)).

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

4: **v**$(ldv,:)$ – REAL (KIND=nag_wp) array

The second dimension of the array **ldv** must be at least $\max(1,\mathbf{k})$.

The m_v by n_v matrix V; this should remain unchanged from the array **b** returned by a previous call to nag_lapack_dtpqrt (f08bb).

5: **t**$(ldt,:)$ – REAL (KIND=nag_wp) array

The first dimension of the array **t** must be at least **nb**.

The second dimension of the array **t** must be at least $\max(1,\mathbf{k})$.

This must remain unchanged from a previous call to nag_lapack_dtpqrt (f08bb) (see **t** in nag_lapack_dtpqrt (f08bb)).

6: **c1**$(ldc1,:)$ – REAL (KIND=nag_wp) array

The first dimension, *ldc1*, of the array **c1** must satisfy

if **side** = 'L', $ldc1 \geq \max(1,\mathbf{k})$;
if **side** = 'R', $ldc1 \geq \max(1,\mathbf{m})$.

The second dimension of the array **c1** must be at least $\max(1,\mathbf{n})$ if **side** = 'L' and at least $\max(1,\mathbf{k})$ if **side** = 'R'.

C_1, the first part of the composite matrix C:

if **side** = 'L'
then **c1** contains the first k rows of C;

if **side** = 'R'
then **c1** contains the first k columns of C.

7: **c2**$(ldc2,:)$ – REAL (KIND=nag_wp) array

The first dimension of the array **c2** must be at least $\max(1,\mathbf{m})$.

The second dimension of the array **c2** must be at least $\max(1,\mathbf{n})$.

C_2, the second part of the composite matrix C.

if **side** = 'L'
then **c2** contains the remaining m_v rows of C;

if **side** = 'R'
then **c2** contains the remaining m_v columns of C;

5.2 Optional Input Parameters

1: **m** – INTEGER

Default: the first dimension of the array **v**.

The number of rows of the matrix C_2, that is,

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

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

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

2: **n** – INTEGER

Default: the first dimension of the array **v**.

The number of columns of the matrix C_2, that is,

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

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

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

3: **k** – INTEGER

Default: the second dimension of the array **t**.

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

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

4: **nb** – INTEGER

Default: the first dimension of the array **t**.

nb, the blocking factor used in a previous call to nag_lapack_dtpqrt (f08bb) to compute the QR factorization of a triangular-pentagonal matrix containing composite matrices A and B.

Constraints:

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

5.3 Output Parameters

1: **c1**($ldc1$, :) – REAL (KIND=nag_wp) array

The first dimension, $ldc1$, of the array **c1** will be

if **side** = 'L', $ldc1 = \max(1, \mathbf{k})$;
if **side** = 'R', $ldc1 = \max(1, \mathbf{m})$.

The second dimension of the array **c1** will be $\max(1, \mathbf{n})$ if **side** = 'L' and at least $\max(1, \mathbf{k})$ if **side** = 'R'.

c1 stores the corresponding block of QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T}.

2: **c2**($ldc2$, :) – REAL (KIND=nag_wp) array

The first dimension of the array **c2** will be $\max(1, \mathbf{m})$.

The second dimension of the array **c2** will be $\max(1, \mathbf{n})$.

c2 stores the corresponding block of QC or $Q^{\mathrm{T}}C$ or CQ or CQ^{T}.

3: **info** – INTEGER

info = 0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

info < 0

If **info** = $-i$, argument i had an illegal value. An explanatory message is output, and execution of the program is terminated.

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 Further Comments

The total number of floating-point operations is approximately $2nk(2m - k)$ if **side** = 'L' and $2mk(2n - k)$ if **side** = 'R'.

The complex analogue of this function is nag_lapack_ztpmqrt (f08bq).

9 Example

See Section 10 in nag_lapack_dtpqrt (f08bb).

9.1 Program Text

```
function f08bc_example

fprintf('f08bc example results\n\n');

% Minimize ||Ax - b|| using recursive QR for m-by-n A and m-by-p B

m = nag_int(6);
n = nag_int(4);
p = nag_int(2);
a = [-0.57, -1.28, -0.39,  0.25;
     -1.93,  1.08, -0.31, -2.14;
```

```
      2.30,  0.24,  0.40, -0.35;
     -1.93,  0.64, -0.66,  0.08;
      0.15,  0.30,  0.15, -2.13;
     -0.02,  1.03, -1.43,  0.50];
b = [-2.67,  0.41;
     -0.55, -3.10;
      3.34, -4.01;
     -0.77,  2.76;
      0.48, -6.17;
      4.10,  0.21];

nb = n;
% Compute the QR Factorisation of first n rows of A
[QRn, Tn, info] = f08ab( ...
  nb,a(1:n,:));

% Compute C = (C1) = (Q^T)*B
[c1, info] = f08ac( ...
     'Left', 'Transpose', QRn, Tn, b(1:n,:));

% Compute least-squares solutions by backsubstitution in R*X = C1
[x, info] = f07te( ...
    'Upper', 'No Transpose', 'Non-Unit', QRn, c1);

% Print first n-row solutions
disp('Solution for n rows');
disp(x(1:n,:));

% Add the remaining rows and perform QR update
nb2 = m-n;
l = nag_int(0);
[R, Q, T, info] = f08bb( ...
  l, nb2, QRn, a(n+1:m,:));

% Apply orthogonal transformations to C
[c1,c2,info] = f08bc( ...
       'Left','Transpose', l, Q, T, c1, b(n+1:m,:));

% Compute least-squares solutions for first n rows: R*X = C1
[x, info] = f07te( ...
    'Upper', 'No transpose', 'Non-Unit', R, c1);
% Print least-squares solutions for all m rows
disp('Least squares solution');
disp(x(1:n,:));

% Compute and print estimates of the square roots of the residual
% sums of squares
for j=1:p
  rnorm(j) = norm(c2(:,j));
end
fprintf('Square roots of the residual sums of squares\n');
fprintf('%12.2e', rnorm);
fprintf('\n');
```

9.2 Program Results

```
     f08bc example results

Solution for n rows
    1.5179   -1.5850
    1.8629    0.5531
   -1.4608    1.3485
    0.0398    2.9619

Least squares solution
    1.5339   -1.5753
    1.8707    0.5559
```

```
   -1.5241    1.3119
    0.0392    2.9585

Square roots of the residual sums of squares
    2.22e-02    1.38e-02
```