

NAG Toolbox

nag_lapack_dormrq (f08ck)

1 Purpose

nag_lapack_dormrq (f08ck) multiplies a general real m by n matrix C by the real orthogonal matrix Q from an RQ factorization computed by nag_lapack_dgerqf (f08ch).

2 Syntax

```
[a, c, info] = nag_lapack_dormrq(side, trans, a, tau, c, 'm', m, 'n', n, 'k', k)
[a, c, info] = f08ck(side, trans, a, tau, c, 'm', m, 'n', n, 'k', k)
```

3 Description

nag_lapack_dormrq (f08ck) is intended to be used following a call to nag_lapack_dgerqf (f08ch), which performs an RQ factorization of a real matrix A and represents the orthogonal matrix Q as a product of elementary reflectors.

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

$$QC, \quad Q^T C, \quad CQ, \quad CQ^T,$$

overwriting the result on C , which may be any real 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_lapack_dgerqf (f08ch).

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>

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd 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: **a**(*lda*, :) – REAL (KIND=nag_wp) array

The first dimension of the array **a** must be at least $\max(1, k)$.

The second dimension of the array **a** must be at least $\max(1, m)$ if **side** = 'L' and at least $\max(1, n)$ if **side** = 'R'.

The *i*th row of **a** must contain the vector which defines the elementary reflector H_i , for $i = 1, 2, \dots, k$, as returned by nag_lapack_dgerqf (f08ch).

4: **tau**(:) – REAL (KIND=nag_wp) array

The dimension of the array **tau** must be at least $\max(1, k)$.

tau(*i*) must contain the scalar factor of the elementary reflector H_i , as returned by nag_lapack_dgerqf (f08ch).

5: **c**(*ldc*, :) – REAL (KIND=nag_wp) array

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

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

The m by n matrix C .

5.2 Optional Input Parameters

1: **m** – INTEGER

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

m , the number of rows of the matrix C .

Constraint: **m** ≥ 0 .

2: **n** – INTEGER

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

n , the number of columns of the matrix C .

Constraint: **n** ≥ 0 .

3: **k** – INTEGER

Default: the first dimension of the array **a** and the dimension of the array **tau**.

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

Constraints:

if **side** = 'L', $m \geq k \geq 0$;

if **side** = 'R', $n \geq k \geq 0$.

5.3 Output Parameters

1: **a**(*lda*, :) – REAL (KIND=nag_wp) array

The first dimension of the array **a** will be $\max(1, k)$.

The second dimension of the array **a** will be $\max(1, m)$ if **side** = 'L' and at least $\max(1, n)$ if **side** = 'R'.

Is modified by nag_lapack_dormrq (f08ck) but restored on exit.

2: **c**(*ldc*,:) – REAL (KIND=nag_wp) array
 The first dimension of the array **c** will be max(1,**m**).
 The second dimension of the array **c** will be max(1,**n**).
 c stores QC or $Q^T C$ or CQ or CQ^T as specified by **side** and **trans**.
 3: **info** – INTEGER
 info = 0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

info = $-i$

If **info** = $-i$, parameter *i* had an illegal value on entry. The parameters are numbered as follows:
 1: **side**, 2: **trans**, 3: **m**, 4: **n**, 5: **k**, 6: **a**, 7: **lda**, 8: **tau**, 9: **c**, 10: **ldc**, 11: **work**, 12: **lwork**, 13: **info**.

It is possible that **info** refers to a parameter that is omitted from the MATLAB interface. This usually indicates that an error in one of the other input parameters has caused an incorrect value to be inferred.

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_zunmrq (f08cx).

9 Example

See Section 10 in nag_lapack_dgerqf (f08ch).

9.1 Program Text

```
function f08ck_example

fprintf('f08ck example results\n\n');

% Find x that minimizes norm(c-Ax) subject to Bx = d .

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 = [1, 0, -1, 0;
      0, 1, 0, -1];
c = [-1.50; -2.14; 1.23; -0.54; -1.68; 0.82];
d = [0; 0];
```

```
% Compute the generalized RQ factorization of (B,A) as
% A = ZRQ, B = TQ
[TQ, taub, ZR, taua, info] = ...
f08zf(b, a);

% Set Qx = y. The problem reduces to:
% minimize (Ry - Z^Tc) subject to Ty = d

% Update c = Z^Tc -> minimize (Ry-c)
[cup, info] = f08ag( ...
    'Left', 'Transpose', ZR, taua, c);

% Solve Ty = d for last p elements
T12 = triu(TQ(1:p, n-p+1:n));

[y2, info] = f07te( ...
    'Upper', 'No transpose', 'Non-unit', T12, d);

% (from Ry-c) R11*y1 + R12*y2 = c1 --> R11*y1 = c1 - R12*y2
% Update c1
c1 = cup(1:n-p) - ZR(1:n-p, n-p+1:n)*y2;

% Solve R11*y1 = c1 for y1
R11 = triu(ZR(1:n-p, 1:n-p));
[y1, info] = f07te( ...
    'Upper', 'No transpose', 'Non-unit', R11, c1);

% Construct y and backtransform for x = Q^Ty
y = [y1; y2];
[~, x, info] = f08ck( ...
    'Left', 'Transpose', TQ, taub, y);
fprintf('Constrained least squares solution\n');
disp(x);

fprintf('Residuals computed directly\n');
res = a*x - c;
disp(res);
fprintf('Residual norm\n');
disp(norm(res));
```

9.2 Program Results

f08ck example results

Constrained least squares solution

```
0.4890
0.9975
0.4890
0.9975
```

Residuals computed directly

```
0.0030
-0.0129
-0.0193
-0.0084
0.0012
-0.0029
```

Residual norm

```
0.0251
```
