

NAG Library Function Document

nag_dgeqlf (f08cec)

1 Purpose

nag_dgeqlf (f08cec) computes a QL factorization of a real m by n matrix A .

2 Specification

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

void nag_dgeqlf (Nag_OrderType order, Integer m, Integer n, double a[],
                Integer pda, double tau[], NagError *fail)
```

3 Description

nag_dgeqlf (f08cec) forms the QL factorization of an arbitrary rectangular real m by n matrix.

If $m \geq n$, the factorization is given by:

$$A = Q \begin{pmatrix} 0 \\ L \end{pmatrix},$$

where L is an n by n lower triangular matrix and Q is an m by m orthogonal matrix. If $m < n$ the factorization is given by

$$A = QL,$$

where L is an m by n lower trapezoidal matrix and Q is again an m by m orthogonal matrix. In the case where $m > n$ the factorization can be expressed as

$$A = (Q_1 \quad Q_2) \begin{pmatrix} 0 \\ L \end{pmatrix} = Q_2 L,$$

where Q_1 consists of the first $m - n$ columns of Q , and Q_2 the remaining n columns.

The matrix Q is not formed explicitly but is represented as a product of $\min(m, n)$ elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction for details). Functions are provided to work with Q in this representation (see Section 9).

Note also that for any $k < n$, the information returned in the last k columns of the array \mathbf{a} represents a QL factorization of the last k columns of the original matrix A .

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 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: **m** – Integer *Input*

On entry: m , the number of rows of the matrix A .

Constraint: $m \geq 0$.

3: **n** – Integer *Input*

On entry: n , the number of columns of the matrix A .

Constraint: $n \geq 0$.

4: **a**[*dim*] – double *Input/Output*

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

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

Where $\mathbf{A}(i, j)$ appears in this document, it refers to the array element

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

On entry: the m by n matrix A .

On exit: if $m \geq n$, the lower triangle of the subarray $\mathbf{A}(m - n + 1 : m, 1 : n)$ contains the n by n lower triangular matrix L .

If $m \leq n$, the elements on and below the $(n - m)$ th superdiagonal contain the m by n lower trapezoidal matrix L . The remaining elements, with the array **tau**, represent the orthogonal matrix Q as a product of elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

5: **pda** – Integer *Input*

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

Constraints:

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

6: **tau**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **tau** must be at least $\max(1, \min(\mathbf{m}, \mathbf{n}))$.

On exit: the scalar factors of the elementary reflectors (see Section 9).

7: **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.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

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

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

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

NE_INT_2

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

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

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.

7 Accuracy

The computed factorization is the exact factorization of a nearby matrix $(A + E)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the *machine precision*.

8 Parallelism and Performance

nag_dgeqlf (f08cec) is not threaded by NAG in any implementation.

nag_dgeqlf (f08cec) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations is approximately $\frac{2}{3}n^2(3m - n)$ if $m \geq n$ or $\frac{2}{3}m^2(3n - m)$ if $m < n$.

To form the orthogonal matrix Q nag_dgeqlf (f08cec) may be followed by a call to nag_dorgql (f08cfc):

```
nag_dorgql(order,m,m,MIN(m,n),&a,pda,tau,&fail)
```

but note that the second dimension of the array **a** must be at least **m**, which may be larger than was required by nag_dgeqlf (f08cec).

When $m \geq n$, it is often only the first n columns of Q that are required, and they may be formed by the call:

```
nag_dorgql(order,m,n,n,&a,pda,tau,&fail)
```

To apply Q to an arbitrary real rectangular matrix C , nag_dgeqlf (f08cec) may be followed by a call to nag_dormql (f08cgc). For example,

```
nag_dormql(order,Nag_LeftSide,Nag_Trans,m,p,MIN(m,n),&a,pda,tau,&c,pdc,&fail)
```

forms $C = Q^T C$, where C is m by p .

The complex analogue of this function is nag_zgeqlf (f08csc).

10 Example

This example solves the linear least squares problems

$$\min_x \|b_j - Ax_j\|_2, \quad j = 1, 2$$

for x_1 and x_2 , where b_j is the j th column of the matrix B ,

$$A = \begin{pmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -2.67 & 0.41 \\ -0.55 & -3.10 \\ 3.34 & -4.01 \\ -0.77 & 2.76 \\ 0.48 & -6.17 \\ 4.10 & 0.21 \end{pmatrix}.$$

The solution is obtained by first obtaining a QL factorization of the matrix A .

10.1 Program Text

```

/* nag_dgeqlf (f08cec) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer      i, j, m, n, nrhs, pda, pdb;
    Integer      exit_status = 0;
    /* Arrays */
    double       *a = 0, *b = 0, *rnorm = 0, *tau = 0;
    /* Nag Types */
    Nag_OrderType order;
    NagError     fail;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J - 1) * pda + I - 1]
#define B(I, J) b[(J - 1) * pdb + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
#define B(I, J) b[(I - 1) * pdb + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_dgeqlf (f08cec) Example Program Results\n\n");

    /* Skip heading in data file */
    scanf("%*[\n]");
    scanf("%ld%ld%ld%*[\n]", &m, &n, &nrhs);

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m*n, double)) ||
        !(b = NAG_ALLOC(m*nrhs, double)) ||

```

```

        !(rnorm = NAG_ALLOC(nrhs, double)) ||
        !(tau = NAG_ALLOC(n, double))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

#ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = m;
#else
    pda = n;
    pdb = nrhs;
#endif

/* Read A and B from data file */
for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j)
        scanf("%lf", &A(i, j));
scanf("%*[\n]");

for (i = 1; i <= m; ++i)
    for (j = 1; j <= nrhs; ++j)
        scanf("%lf", &B(i, j));
scanf("%*[\n]");

/* nag_dgeqlf (f08cec).
 * Compute the QL factorization of A.
 */
nag_dgeqlf(order, m, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgeqlf (f08cec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dormql (f08cgc).
 * Compute C = (C1) = (Q**T)*B, storing the result in B.
 *          (C2)
 */
nag_dormql(order, Nag_LeftSide, Nag_Trans, m, nrhs, n, a, pda, tau, b, pdb,
           &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dormql (f08cgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dtrtrs (f07tec).
 * Compute least-squares solutions by backsubstitution in
 * L*X = C2.
 */
nag_dtrtrs(order, Nag_Lower, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs,
           &A(m - n + 1, 1), pda, &B(m - n + 1, 1), pdb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dtrtrs (f07tec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_gen_real_mat_print (x04cac).
 * Print least-squares solution(s).
 */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs,
                       &B(m - n + 1, 1), pdb, "Least-squares solution(s)",
                       0, &fail);

```

```

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dge_norm (f16rac).
 * Compute and print estimates of the square roots of the residual
 * sums of squares.
 */
for (j = 1; j <= nrhs; ++j) {
    nag_dge_norm(order, Nag_FrobeniusNorm, m-n, 1, &B(1, j), pdb,
                 &rnorm[j-1], &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

printf("\nSquare root(s) of the residual sum(s) of squares\n");
for (j = 0; j < nrhs; ++j)
    printf("%11.2e%s", rnorm[j], (j+1)%7 == 0?"\n":" ");

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(rnorm);
NAG_FREE(tau);

return exit_status;
}

#undef A
#undef B

```

10.2 Program Data

nag_dgeqlf (f08cec) Example Program Data

```

6      4      2      :Values of m, n and nrhs

-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 :End of matrix A

-2.67   0.41
-0.55  -3.10
 3.34  -4.01
-0.77   2.76
 0.48  -6.17
 4.10   0.21      :End of matrix B

```

10.3 Program Results

nag_dgeqlf (f08cec) Example Program Results

Least-squares solution(s)

	1	2
1	1.5339	-1.5753
2	1.8707	0.5559
3	-1.5241	1.3119
4	0.0392	2.9585

Square root(s) of the residual sum(s) of squares

2.22e-02	1.38e-02
----------	----------
