

# NAG Library Function Document

## nag\_zgeqlf (f08csc)

### 1 Purpose

nag\_zgeqlf (f08csc) computes a  $QL$  factorization of a complex  $m$  by  $n$  matrix  $A$ .

### 2 Specification

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

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

### 3 Description

nag\_zgeqlf (f08csc) forms the  $QL$  factorization of an arbitrary rectangular complex  $m$  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  $m$  unitary 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  $m$  unitary 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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*] – Complex *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 unitary 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*] – Complex *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 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

**NE\_BAD\_PARAM**

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

**NE\_INT**

On entry,  $\mathbf{m} = \langle value \rangle$ .

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

On entry,  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

**NE\_INT\_2**

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{m} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} \geq \max(1, \mathbf{m})$ .

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .

**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.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 3.6.6 in How to Use the NAG Library and its Documentation 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 How to Use the NAG Library and its Documentation for further information.

**7 Accuracy**

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

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

and  $\epsilon$  is the *machine precision*.

**8 Parallelism and Performance**

nag\_zgeqlf (f08csc) 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 real floating-point operations is approximately  $\frac{8}{3}n^2(3m - n)$  if  $m \geq n$  or  $\frac{8}{3}m^2(3n - m)$  if  $m < n$ .

To form the unitary matrix  $Q$  nag\_zgeqlf (f08csc) may be followed by a call to nag\_zungql (f08ctc):

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

but note that the second dimension of the array  $\mathbf{a}$  must be at least  $\mathbf{m}$ , which may be larger than was required by nag\_zgeqlf (f08csc).

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

To apply  $Q$  to an arbitrary complex rectangular matrix  $C$ , `nag_zgeqlf` (f08csc) may be followed by a call to `nag_zunmql` (f08cuc). For example,

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

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

The real analogue of this function is `nag_dgeqlf` (f08cec).

## 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  $B$ ,

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\ -0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\ 0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -2.09 + 1.93i & 3.26 - 2.70i \\ 3.34 - 3.53i & -6.22 + 1.16i \\ -4.94 - 2.04i & 7.94 - 3.13i \\ 0.17 + 4.23i & 1.04 - 4.26i \\ -5.19 + 3.63i & -2.31 - 2.12i \\ 0.98 + 2.53i & -1.39 - 4.05i \end{pmatrix}.$$

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

### 10.1 Program Text

```
/* nag_zgeqlf (f08csc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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 */
    Complex *a = 0, *b = 0, *tau = 0;
    double *rnorm = 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_zgeqlf (f08csc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n]", &m, &n, &nrhs);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n]", &m, &n, &nrhs);
#endif

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

/* Allocate memory */
if (!(a = NAG_ALLOC(m * n, Complex)) ||
    !(b = NAG_ALLOC(m * nrhs, Complex)) ||
    !(tau = NAG_ALLOC(n, Complex)) || !(rnorm = NAG_ALLOC(nrhs, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A and B from data file */
for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

    for (i = 1; i <= m; ++i)
        for (j = 1; j <= nrhs; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");

```

```

#else
    scanf("%*[\n]");
#endif

/* nag_zgeqlf (f08csc).
 * Compute the QL factorization of A.
 */
nag_zgeqlf(order, m, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgeqlf (f08csc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zunmql (f08cuc).
 * Compute  $C = (C1) = (Q^H) * B$ , storing the result in B.
 *      (C2)
 */
nag_zunmql(order, Nag_LeftSide, Nag_ConjTrans, m, nrhs, n, a, pda, tau,
           b, pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zunmql (f08cuc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_ztrtrs (f07tsc).
 * Compute least squares solutions by back-substitution in
 *  $L * X = C2$ .
 */
nag_ztrtrs(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_ztrtrs (f07tsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print least squares solution(s).
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                             nrhs, &B(m - n + 1, 1), pdb, Nag_BracketForm,
                             "%7.4f", "Least squares solution(s)",
                             Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                             80, 0, 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zge_norm (f16uac).
 * Compute and print estimates of the square roots of the residual
 * sums of squares.
 */
for (j = 1; j <= nrhs; ++j) {
    nag_zge_norm(order, Nag_FrobeniusNorm, m - n, 1, &B(1, j), pdb,
                &rnorm[j - 1], &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_zge_norm (f16uac).\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(tau);
  NAG_FREE(rnorm);

  return exit_status;
}

#undef A
#undef B

```

## 10.2 Program Data

```

nag_zgeqlf (f08csc) Example Program Data
  6           4           2           :Values of M, N and NRHS

( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06) (-0.05, 0.41)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42) (-0.81, 0.56)
( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17) (-1.11, 0.60)
(-0.37, 0.38) ( 0.19,-0.54) (-0.98,-0.36) ( 0.22,-0.20)
( 0.83, 0.51) ( 0.20, 0.01) (-0.17,-0.46) ( 1.47, 1.59)
( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23) ( 0.26, 0.26) :End of matrix A

(-2.09, 1.93) ( 3.26,-2.70)
( 3.34,-3.53) (-6.22, 1.16)
(-4.94,-2.04) ( 7.94,-3.13)
( 0.17, 4.23) ( 1.04,-4.26)
(-5.19, 3.63) (-2.31,-2.12)
( 0.98, 2.53) (-1.39,-4.05) :End of matrix B

```

## 10.3 Program Results

```

nag_zgeqlf (f08csc) Example Program Results

Least squares solution(s)
  1           2
1 (-0.5044,-1.2179) ( 0.7629, 1.4529)
2 (-2.4281, 2.8574) ( 5.1570,-3.6089)
3 ( 1.4872,-2.1955) (-2.6518, 2.1203)
4 ( 0.4537, 2.6904) (-2.7606, 0.3318)

Square root(s) of the residual sum(s) of squares
  6.88e-02   1.87e-01

```

---