NAG Library Function Document
nag_zgeqlf (f08csc)

1 Purpose
nag_zgeqlf (f08csc) computes a {\it QL} factorization of a complex {\it m} by {\it n} matrix {\it A}.

2 Specification
```c
#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 {\it QL} factorization of an arbitrary rectangular complex {\it m} by {\it n} matrix.
If {\it m} \geq {\it n}, the factorization is given by:

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

where {\it L} is an {\it n} by {\it n} lower triangular matrix and {\it Q} is an {\it m} by {\it m} unitary matrix. If {\it m} < {\it n} the factorization is given by

\[ A = QL, \]

where {\it L} is an {\it m} by {\it n} lower trapezoidal matrix and {\it Q} is again an {\it m} by {\it m} unitary matrix. In the case where {\it m} > {\it n} the factorization can be expressed as

\[ A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} 0 \\
L
\end{pmatrix} = Q_2L, \]

where {\it Q}_1 consists of the first {\it m} – {\it n} columns of {\it Q}, and {\it Q}_2 the remaining {\it n} columns.

The matrix {\it 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 {\it Q} in this representation (see Section 9).

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

4 References

5 Arguments
1: \textbf{order} – Nag_OrderType

\textit{Input}

On entry: the \textbf{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
\textbf{order} = \texttt{Nag\_RowMajor}. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint: order = Nag\_RowMajor or Nag\_ColMajor.}

2: \textbf{m} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \texttt{m}, the number of rows of the matrix \texttt{A}.

\textit{Constraint:} \texttt{m} \geq 0.

3: \textbf{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \texttt{n}, the number of columns of the matrix \texttt{A}.

\textit{Constraint:} \texttt{n} \geq 0.

4: \textbf{a}[\text{dim}] – Complex \hspace{1cm} \textit{Input/Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{a} must be at least

\[
\max(1, \texttt{pda} \times \texttt{n}) \text{ when order = Nag\_ColMajor;}
\]

\[
\max(1, \texttt{m} \times \texttt{pda}) \text{ when order = Nag\_RowMajor.}
\]

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

\[
a[(j - 1) \times \texttt{pda} + i - 1] \text{ when order = Nag\_ColMajor;}
\]

\[
a[(i - 1) \times \texttt{pda} + j - 1] \text{ when order = Nag\_RowMajor.}
\]

\textit{On entry:} the \texttt{m} by \texttt{n} matrix \texttt{A}.

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

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

5: \textbf{pda} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \textbf{order}) in the array \texttt{a}.

\textit{Constraints:}

\[
\text{if order = Nag\_ColMajor, \texttt{pda} \geq \max(1, \texttt{m});}
\]

\[
\text{if order = Nag\_RowMajor, \texttt{pda} \geq \max(1, \texttt{n}).}
\]

6: \textbf{tau}[\text{dim}] – Complex \hspace{1cm} \textit{Output}

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

\textit{On exit:} the scalar factors of the elementary reflectors (see Section 9).

7: \textbf{fail} – NagError * \hspace{1cm} \textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \textbf{Error Indicators and Warnings}

\textbf{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 \( \langle \text{value} \rangle \) had an illegal value.

NE_INT
On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m \geq 0 \).
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).
On entry, \( pda = \langle \text{value} \rangle \).
Constraint: \( pda > 0 \).

NE_INT_2
On entry, \( pda = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, m) \).
On entry, \( pda = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, n) \).

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 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 factorization is the exact factorization of a nearby matrix \( (A + E) \), where
\[
\|E\|_2 = O(\epsilon) \|A\|_2,
\]
and \( \epsilon \) is the \textit{machine precision}.

8 Parallelism and Performance
\texttt{nag\_zgeqlf (f08csc)} is not threaded by NAG in any implementation.
\texttt{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 \( \frac{2}{3}n^2(3m - n) \) if \( m \geq n \) or \( \frac{2}{3}n^2(3n - m) \) if \( m < n \).
To form the unitary matrix \( Q \) \texttt{nag\_zgeqlf (f08csc)} may be followed by a call to \texttt{nag\_zungql (f08ctc)}:
\[
nag\_zungql(order,m,m,\min(m,n),\&a,pda,tau,\&fail)
\]
but note that the second dimension of the array \( a \) must be at least \( m \), which may be larger than was required by nag_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:

\[
\text{nag_zunqql(order,m,\(n\),&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,

\[
\text{nag_zunmql(order,Nag_LeftSide,Nag_ConjTrans,m,p,MIN(m,n),&a,pda,tau,&c,pdc,&fail)}
\]

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. */
/* Copyright 2014 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 */
    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]
#else
#define A(I, J) a[(I - 1) * pda + J - 1]
#define B(I, J) b[(I - 1) * pdb + J - 1]
#endif

INIT_FAIL(fail);
printf("nag_zgeqlf (f08csc) Example Program Results\n\n");
/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\r\n]");
#else
scanf("%*[\r\n]");
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\r\n]", &m, &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\r\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("%*[\r\n]");
    #else
    scanf("%*[\r\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("%*[\r\n]");
    #else
    scanf("%*[\r\n]");
    #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", 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", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_ztrtrs (f07tsc). * Compute least-squares solutions by backsubstitution 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", 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(1, 1), pdb, Nag_BracketForm,
                              "%7.4f", "Least-squares solution(s)",
                              Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
                              &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\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", 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

Values of M, N and NRHS
6 4 2
( 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)
( -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 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