NAG C Library Function Document

nag_zgeqrf (f08asc)

1 Purpose

nag_zgeqrf (f08asc) computes the QR factorization of a complex \( m \times n \) matrix.

2 Specification

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

3 Description

nag_zgeqrf (f08asc) forms the QR factorization of an arbitrary rectangular complex \( m \times n \) matrix. No pivoting is performed.

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

\[
A = QR
\]

where \( R \) is an \( n \times n \) upper triangular matrix (with real diagonal elements) and \( Q \) is an \( m \times m \) unitary matrix. It is sometimes more convenient to write the factorization as:

\[
A = (Q_1 \quad Q_2) \begin{pmatrix} R \cr 0 \end{pmatrix}
\]

which reduces to

\[
A = Q_1 R,
\]

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

If \( m < n \), \( R \) is trapezoidal, and the factorization can be written

\[
A = Q(R_1 \quad R_2),
\]

where \( R_1 \) is upper triangular and \( R_2 \) is rectangular.

The matrix \( Q \) is not formed explicitly but is represented as a product of \( \min(m, n) \) elementary reflectors (see the f08 Chapter Introduction for details). Functions are provided to work with \( Q \) in this representation (see Section 8).

Note also that for any \( k < n \), the information returned in the first \( k \) columns of the array \( a \) represents a QR factorization of the first \( k \) columns of the original matrix \( A \).

4 References


5 Parameters

1:  order – Nag_OrderType  

\textit{Input}

\textit{On entry:} the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

\textit{Constraint:} order = Nag_RowMajor or Nag_ColMajor.
2: \( m \) – Integer \hspace{1cm} \text{Input}

\textit{On entry}: \( m \), the number of rows of the matrix \( A \).
\textit{Constraint}: \( m \geq 0 \).

3: \( n \) – Integer \hspace{1cm} \text{Input}

\textit{On entry}: \( n \), the number of columns of the matrix \( A \).
\textit{Constraint}: \( n \geq 0 \).

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

\textit{Note}: the dimension, \( \text{dim} \), of the array \( a \) must be at least \( \max(1, \text{pda} \times n) \) when \( \text{order} = \text{Nag\_ColMajor} \) and at least \( \max(1, \text{pda} \times m) \) when \( \text{order} = \text{Nag\_RowMajor} \).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(j-1) \times \text{pda} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( a[(i-1) \times \text{pda} + j - 1] \).

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

\textit{On exit}: if \( m \geq n \), the elements below the diagonal are overwritten by details of the unitary matrix \( Q \) and the upper triangle is overwritten by the corresponding elements of the \( n \) by \( n \) upper triangular matrix \( R \).

If \( m < n \), the strictly lower triangular part is overwritten by details of the unitary matrix \( Q \) and the remaining elements are overwritten by the corresponding elements of the \( m \) by \( n \) upper trapezoidal matrix \( R \).

The diagonal elements of \( R \) are real.

5: \( \text{pda} \) – Integer \hspace{1cm} \text{Input}

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

\textit{Constraints}:

\begin{align*}
\text{if} & \ \text{order} = \text{Nag\_ColMajor}, & \text{pda} & \geq \max(1, m); \\
\text{if} & \ \text{order} = \text{Nag\_RowMajor}, & \text{pda} & \geq \max(1, n).
\end{align*}

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

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

\textit{On exit}: further details of the unitary matrix \( Q \).

7: \( \text{fail} \) – NagError * \hspace{1cm} \text{Output}

The NAG error parameter (see the Essential Introduction).

6 \hspace{0.5cm} \textbf{Error Indicators and Warnings}

\textbf{NE\_INT}

\textit{On entry}, \( m = \langle \text{value} \rangle \).
\textit{Constraint}: \( m \geq 0 \).

\textit{On entry}, \( n = \langle \text{value} \rangle \).
\textit{Constraint}: \( n \geq 0 \).

\textit{On entry}, \( \text{pda} = \langle \text{value} \rangle \).
\textit{Constraint}: \( \text{pda} > 0 \).
On entry, \( pda = \langle \text{value} \rangle, m = \langle \text{value} \rangle \).
Constraint: \( pda \geq \max(1, m) \).

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

**NE_ALLOC_FAIL**
Memory allocation failed.

**NE_BAD_PARAM**
On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

**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 consult 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 \( \epsilon \) is the *machine precision*.

### 8 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 \( Q \) this function may be followed by a call to \texttt{nag_zungqr} (f08atc):
\[
\texttt{nag_zungqr}(\text{order},m,m,\text{MIN}(m,n),&a,pda,\tau,&\text{fail})
\]
but note that the second dimension of the array \( a \) must be at least \( m \), which may be larger than was required by \texttt{nag_zgeqrf} (f08asc).

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:
\[
\texttt{nag_zungqr}(\text{order},m,n,n,&a,pda,\tau,&\text{fail})
\]
To apply \( Q \) to an arbitrary complex rectangular matrix \( C \), this function may be followed by a call to \texttt{nag_zunmqr} (f08auc).
\[
\texttt{nag_zunmqr}(\text{order},\text{Nag_LeftSide},\text{Nag_ConjTrans},m,p,\text{MIN}(m,n),&a,pda,\tau,c,c,\text{pdc},&\text{fail})
\]
forms \( C = Q^H C \), where \( C \) is \( m \) by \( p \).

To compute a \( QR \) factorization with column pivoting, use \texttt{nag_zgeqpf} (f08bsc).

The real analogue of this function is \texttt{nag_dgeqrf} (f08aec).

### 9 Example

To solve the linear least-squares problem
\[
\minimize \| Ax_i - b_i \|_2, \quad i = 1, 2
\]
where \( b_1 \) and \( b_2 \) are the columns 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

\[
B = \begin{pmatrix}
-1.54 + 0.76i & 3.17 - 2.09i \\
0.12 - 1.92i & -6.53 + 4.18i \\
-9.08 - 4.31i & 7.28 + 0.73i \\
7.49 + 3.65i & 0.91 - 3.97i \\
-5.63 - 2.12i & -5.46 - 1.64i \\
2.37 + 8.03i & -2.84 - 5.86i
\end{pmatrix}
\]

### 9.1 Program Text

/* nag_zgeqrf (f08asc) Example Program.  
    * Copyright 2001 Numerical Algorithms Group.  
    */

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

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

    #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);
    Vprintf("f08asc Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%*[\n"]");
    Vscanf("%ld%ld%ld%*[\n"] , &m, &n, &nrhs);

    #ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = m;
    #else
    pda = n;
    pdb = nrhs;
    #endif
    tau_len = MIN(m,n);
}
/* Allocate memory */
if ( !(a = NAG_ALLOC(m * n, Complex)) ||
    !(b = NAG_ALLOC(m * nrhs, Complex)) ||
    !(tau = NAG_ALLOC(tau_len, Complex)) )
{
    Vprintf("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)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
}
Vscanf("%*[\n ]");
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
}
Vscanf("%*[\n ]");

/* Compute the QR factorization of A */
f08asc(order, m, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08asc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute C = (Q**H)*B, storing the result in B */
f08auc(order, Nag_LeftSide, Nag_ConjTrans, m, nrhs, n, a, pda,
        tau, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08auc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute least-squares solution by backsubstitution in R*X = C */
f07tsc(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs,
        a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07tsc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print least-squares solution(s) */
Vprintf("\n");
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb,
        Nag_BracketForm, "%7.4f", "Least-squares solution(s)",
        Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (tau) NAG_FREE(tau);
return exit_status;
}
9.2 Program Data

f08asc Example Program Data

Values of M, N and NRHS

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>NRHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

```
( 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)
( 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)
( 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)
```

9.3 Program Results

f08asc Example Program Results

Least-squares solution(s)

```
1  1 (-0.4936, -1.1993) ( 0.7535, 1.4404)
2 (-2.4708, 2.8373) ( 5.1726, -3.6235)
3 ( 1.5060, -2.1830) (-2.6609, 2.1334)
4 ( 0.4459, 2.6848) (-2.6966, 0.2711)
```