NAG C Library Function Document

nag_zgeqpf (f08bsc)

1 Purpose

nag_zgeqpf (f08bsc) computes the QR factorization, with column pivoting, of a complex m by n matrix.

2 Specification

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

3 Description

nag_zgeqpf (f08bsc) forms the QR factorization with column pivoting of an arbitrary rectangular complex m by n matrix.

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

\[
AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}
\]

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

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

which reduces to

\[
AP = Q_1 R,
\]

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

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

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 \( \text{min}(m, n) \) 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 permuted matrix \( AP \).

The function allows specified columns of \( A \) to be moved to the leading columns of \( AP \) at the start of the factorization and fixed there. The remaining columns are free to be interchanged so that at the \( i \)-th stage the pivot column is chosen to be the column which maximizes the 2-norm of elements \( i \) to \( m \) over columns \( i \) to \( n \).

4 References


[NP3645/7]
5 Parameters

1: \(\text{order} - \) Nag_OrderType \hspace{1cm} \text{Input}

*On entry:* the \textit{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 \textit{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:} \textit{order} = Nag_RowMajor or Nag_ColMajor.

2: \(m - \) Integer \hspace{1cm} \text{Input}

*On entry:* \(m\), the number of rows of the matrix \(A\).

\textit{Constraint:} \(m \geq 0\).

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

*On entry:* \(n\), the number of columns of the matrix \(A\).

\textit{Constraint:} \(n \geq 0\).

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

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

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

*On entry:* the \(m\) by \(n\) matrix \(A\).

*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: \(pda - \) Integer \hspace{1cm} \text{Input}

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

\textit{Constraints:}

- if \(\text{order} = \) Nag_ColMajor, \(pda \geq \max(1, m)\);
- if \(\text{order} = \) Nag_RowMajor, \(pda \geq \max(1, n)\).

6: \(jpvt[dim] - \) Integer \hspace{1cm} \text{Input/Output}

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

*On entry:* if \(jpvt[i] \neq 0\), then the \(i\)th column of \(A\) is moved to the beginning of \(AP\) before the decomposition is computed and is fixed in place during the computation. Otherwise, the \(i\)th column of \(A\) is a free column (i.e., one which may be interchanged during the computation with any other free column).

*On exit:* details of the permutation matrix \(P\). More precisely, if \(jpvt[i - 1] = k\), then the \(k\)th column of \(A\) is moved to become the \(i\)th column of \(AP\); in other words, the columns of \(AP\) are the columns of \(A\) in the order \(jpvt[0], jpvt[1], \ldots, jpvt[n - 1]\).
7: \[ \text{tau}[\text{dim}] \text{ – Complex} \]

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

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

8: \[ \text{fail} \text{ – NagError *} \]

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

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, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).

NE_INT_2

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

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

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 nag_zungqr (f08atc):
\[
\text{nag_zungqr} (\text{order}, m, m, \text{MIN}(m, n), \&a, \text{pda}, \text{tau}, \&\text{fail})
\]
but note that the second dimension of the array \( a \) must be at least \( m \), which may be larger than was
required by nag_zgeqpf (f08bfc).
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:

\[
\text{nagzungqr (order,m,n,\&a,pda,tau,\&fail)}
\]

To apply \( Q \) to an arbitrary complex rectangular matrix \( C \), this function may be followed by a call to \text{nag_zunmqr (f08auc)}. For example,

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

forms \( C = Q^H C \), where \( C \) is \( m \) by \( p \).

To compute a QR factorization without column pivoting, use \text{nag_zgeqrf (f08asc)}.

The real analogue of this function is \text{nag_dgeqpf (f08bec)}.

9 Example

To solve the linear least-squares problem

\[
\text{minimize} \| Ax_i - b_i \|_2, \quad i = 1, 2
\]

where \( b_1 \) and \( b_2 \) are the columns of the matrix \( B \),

\[
A = \begin{pmatrix}
0.47 - 0.34i & -0.40 + 0.54i & 0.60 + 0.01i & 0.80 - 1.02i \\
-0.32 - 0.23i & -0.05 + 0.20i & -0.26 - 0.44i & -0.43 + 0.17i \\
0.35 - 0.60i & -0.52 - 0.34i & 0.87 - 0.11i & -0.34 - 0.09i \\
0.89 + 0.71i & -0.45 - 0.45i & -0.02 - 0.57i & 1.14 - 0.78i \\
-0.19 + 0.06i & 0.11 - 0.85i & 1.44 + 0.80i & 0.07 + 1.14ik
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix}
-0.85 - 1.63i & 2.49 + 4.01i \\
-2.16 + 3.52i & -0.14 + 7.98i \\
4.57 - 5.71i & 8.36 - 0.28i \\
6.38 - 7.40i & -3.55 + 1.29i \\
8.41 + 9.39i & -6.72 + 5.03ik
\end{pmatrix}
\]

Here \( A \) is approximately rank-deficient, and hence it is preferable to use \text{nag_zgeqpf (f08bsc)} rather than \text{nag_zgeqrf (f08asc)}.

9.1 Program Text

/* nag_zgeqpf (f08bsc) Example Program. *
 * Copyright 2001 Numerical Algorithms Group.
 * *
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double tol;
    Integer i, j, jpvt_len, k, m, n, nrhs;
    Integer pda, pdb, pdx, tau_len;
    NagError exit_status=0;
    NagError fail;

    ...
Nag_OrderType order;
/* Arrays */
Complex *a=0, *b=0, *tau=0, *x=0;
Integer *jpvt=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]
#define X(I,J) x[(J-1)*pdx + I - 1]
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
#define X(I,J) x[(I-1)*pdx + J - 1]
#endif
order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
#define B(I,J) b[(I-1)*pdb + J - 1]
#define X(I,J) x[(I-1)*pdx + J - 1]
#endif
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
Vprintf("f08bsc Example Program Results\n\n");

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

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

ten_len = MIN(m,n);
jpvt_len = n;

/* Allocate memory */
if ( !(a = NAG_ALLOC(m * n, Complex)) ||
!(b = NAG_ALLOC(m * nrhs, Complex)) ||
!(tau = NAG_ALLOC(tau_len, Complex)) ||
!(x = NAG_ALLOC(m * nrhs, Complex)) ||
!(jpvt = NAG_ALLOC(jpvt_len, Integer)) )
{
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] ");

/* Initialize JPVT to be zero so that all columns are free */
f16dbc(n, 0, jpvt, 1, &fail);
/* Compute the QR factorization of A */
f08bsc(order, m, n, a, pda, jpvt, tau, &fail);
if (fail.code != NE_NOERROR)
{
Vprintf("Error from f08bsc.\n\n", fail.message);
exit_status = 1;
goto END;
}
/* Choose TOL to reflect the relative accuracy of the input data */
tol = 0.01;

/* Determine which columns of R to use */
for (k = 1; k <= n; ++k)
  { if (a02dbc(A(k, k)) <= tol * a02dbc(A(1, 1)))
      break;
  }
--k;

/* 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%\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute least-squares solution by backsubstitution in R*B = C */
f07tsc(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, k, nrhs,
       a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR)
  { Vprintf("Error from f07tsc.\n%\n", fail.message);
    exit_status = 1;
    goto END;
  }
for (i = k + 1; i <= n; ++i)
  { for (j = 1; j <= nrhs; ++j)
      { B(i,j).re = 0.0;
        B(i,j).im = 0.0;
      }
  }

/* Unscramble the least-squares solution stored in B */
for (i = 1; i <= n; ++i)
  { for (j = 1; j <= nrhs; ++j)
      { X(jpvt[i - 1], j).re = B(i, j).re;
        X(jpvt[i - 1], j).im = B(i, j).im;
      }
  }

/* Print least-squares solution */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
       Nag_BracketForm, "%7.4f", "Least-squares solution",
       Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)
  { Vprintf("Error from x04dbc.\n%\n", fail.message);
    exit_status = 1;
    goto END;
  }

END:
if (a) NAG_FREE(a);
if (b) NAG_FREE(b);
if (tau) NAG_FREE(tau);
if (x) NAG_FREE(x);
if (jpvt) NAG_FREE(jpvt);
return exit_status;
### 9.2 Program Data

**f08bsc Example Program Data**

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.47, -0.34</td>
<td>-0.40, 0.54</td>
<td>0.60, 0.01</td>
</tr>
<tr>
<td>-0.32, -0.23</td>
<td>-0.05, 0.20</td>
<td>-0.26, -0.44</td>
</tr>
<tr>
<td>0.35, -0.60</td>
<td>-0.52, -0.34</td>
<td>0.87, -0.11</td>
</tr>
<tr>
<td>0.89, 0.71</td>
<td>-0.45, -0.45</td>
<td>-0.02, -0.57</td>
</tr>
<tr>
<td>-0.19, 0.06</td>
<td>0.11, -0.85</td>
<td>1.44, 0.80</td>
</tr>
</tbody>
</table>

:End of matrix A

| -0.85, -1.63 | 2.49, 4.01 |
| -2.16, 3.52 | -0.14, 7.98 |
| 4.57, -5.71 | 8.36, -0.28 |
| 6.38, -7.40 | -3.55, 1.29 |
| 8.41, 9.39 | -6.72, 5.03 |

:End of matrix B

### 9.3 Program Results

**f08bsc Example Program Results**

Least-squares solution

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.0000, 0.0000)</td>
<td>(0.0000, 0.0000)</td>
</tr>
<tr>
<td>(2.6925, 8.0446)</td>
<td>(-2.0563, -2.9759)</td>
</tr>
<tr>
<td>(2.7602, 2.5455)</td>
<td>(1.0588, 1.4635)</td>
</tr>
<tr>
<td>(2.7383, 0.5123)</td>
<td>(-1.4150, 0.2982)</td>
</tr>
</tbody>
</table>