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

$$AP = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}\begin{pmatrix} R \\ 0 \end{pmatrix}$$

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

$$AP = Q\begin{pmatrix} R_1 & R_2 \end{pmatrix},$$

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 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 ith 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

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

1: **order** – Nag_OrderType *Input*

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.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **m** – Integer *Input*

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

Constraint: **m** ≥ 0.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 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** and at least $\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

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

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 *Input*

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

Constraints:

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

6: **jpvt**[dim] – Integer *Input/Output*

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

On entry: if $\mathbf{jpvt}[i] \neq 0$, then the ith 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 ith 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 $\mathbf{jpvt}[i-1] = k$, then the kth column of A is moved to become the ith column of AP; in other words, the columns of AP are the columns of A in the order $\mathbf{jpvt}[0], \mathbf{jpvt}[1], \ldots, \mathbf{jpvt}[n-1]$.

7: **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: further details of the unitary matrix Q.

8: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

NE_INT_2

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ 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 ϵ 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):

```
nag_zungqr (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_zgeqpf (f08bsc).

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

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

```
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 nag_zgeqrf (f08asc).

The real analogue of this function is 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 nag_zgeqpf (f08bsc) rather than nag_zgeqrf (f08asc).

9.1 Program Text

```
/* nag_zgeqpf (f08bsc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#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;
  Integer  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]
  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]
  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

  tau_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%s\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%s\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%s\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%s\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
  5  4  2                                                    :Values of M, N and NRHS
 ( 0.47,-0.34) (-0.40, 0.54) ( 0.60, 0.01) ( 0.80,-1.02)
 (-0.32,-0.23) (-0.05, 0.20) (-0.26,-0.44) (-0.43, 0.17)
 ( 0.35,-0.60) (-0.52,-0.34) ( 0.87,-0.11) (-0.34,-0.09)
 ( 0.89, 0.71) (-0.45,-0.45) (-0.02,-0.57) ( 1.14,-0.78)
 (-0.19, 0.06) ( 0.11,-0.85) ( 1.44, 0.80) ( 0.07, 1.14)   :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
                    1                  2
 1  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)
 2  ( 2.6925, 8.0446)  (-2.0563,-2.9759)
 3  ( 2.7602, 2.5455)  ( 1.0588, 1.4635)
 4  ( 2.7383, 0.5123)  (-1.4150, 0.2982)
```