NAG C Library Function Document

nag_zgelqf (f08avc)

1 Purpose

nag_zgelqf (f08avc) computes the LQ factorization of a complex m by n matrix.

2 Specification

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

3 Description

nag_zgelqf (f08avc) forms the LQ factorization of an arbitrary rectangular complex m by n matrix. No pivoting is performed.

If $m \leq n$, the factorization is given by:

$$A = (L \quad 0) Q$$

where L is an m by m lower triangular matrix (with real diagonal elements) and Q is an n by n unitary matrix. It is sometimes more convenient to write the factorization as

$$A = (L \quad 0) \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}$$

which reduces to

$$A = LQ_1,$$

where Q_1 consists of the first m rows of Q, and Q_2 the remaining $n - m$ rows.

If $m > n$, L is trapezoidal, and the factorization can be written

$$A = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} Q$$

where L_1 is lower triangular and L_2 is rectangular.

The LQ factorization of A is essentially the same as the QR factorization of A^H, since

$$A = (L \quad 0) Q \Leftrightarrow A^H = Q^H \begin{pmatrix} L^H \\ 0 \end{pmatrix}.$$

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 $k < m$, the information returned in the first k rows of the array **a** represents an LQ factorization of the first k rows of the original matrix A.

4 References

None.

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 $\max(1, \textbf{pda} \times \textbf{n})$ when **order** $=$ **Nag_ColMajor** and at least $\max(1, \textbf{pda} \times \textbf{m})$ when **order** $=$ **Nag_RowMajor**.

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

On entry: the m by n matrix A.

On exit: if $m \leq n$, the elements above the diagonal are overwritten by details of the unitary matrix Q and the lower triangle is overwritten by the corresponding elements of the m by m lower triangular matrix L.

If $m > n$, the strictly upper 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 lower trapezoidal matrix L.

The diagonal elements of L 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** $\geq \max(1, \textbf{m})$;
if **order** $=$ **Nag_RowMajor**, **pda** $\geq \max(1, \textbf{n})$.

6: **tau**[dim] – Complex *Output*

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

On exit: further details of the unitary matrix Q.

7: **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** = $\langle value \rangle$.
Constraint: **pda** > 0.

NE_INT_2

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

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ 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 ϵ is the ***machine precision***.

8 Further Comments

The total number of real floating-point operations is approximately $\frac{8}{3}m^2(3n - m)$ if $m \leq n$ or $\frac{8}{3}n^2(3m - n)$ if $m > n$.

To form the unitary matrix Q this function may be followed by a call to nag_zunglq (f08awc):

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

but note that the first dimension of the array **a**, specified by the parameter **pda**, must be at least **n**, which may be larger than was required by nag_zgelqf (f08avc).

When $m \leq n$, it is often only the first m rows of Q that are required, and they may be formed by the call:

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

To apply Q to an arbitrary complex rectangular matrix C, this function may be followed by a call to nag_zunmlq (f08axc). For example,

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

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

The real analogue of this function is nag_dgelqf (f08ahc).

9 Example

To find the minimum-norm solutions of the under-determined systems of linear equations

$$Ax_1 = b_1 \quad \text{and} \quad Ax_2 = b_2$$

where b_1 and b_2 are the columns of the matrix B,

$$A = \begin{pmatrix} 0.28 - 0.36i & 0.50 - 0.86i & -0.77 - 0.48i & 1.58 + 0.66i \\ -0.50 - 1.10i & -1.21 + 0.76i & -0.32 - 0.24i & -0.27 - 1.15i \\ 0.36 - 0.51i & -0.07 + 1.33i & -0.75 + 0.47i & -0.08 + 1.01i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -1.35 + 0.19i & 4.83 - 2.67i \\ 9.41 - 3.56i & -7.28 + 3.34i \\ -7.57 + 6.93i & 0.62 + 4.53i \end{pmatrix}.$$

9.1 Program Text

```
/* nag_zgelqf (f08avc) 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 <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("f08avc 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 = n;
#else
  pda = n;
  pdb = nrhs;
#endif

  tau_len = MIN(m,n);

  /* Allocate memory */
  if ( !(a = NAG_ALLOC(m * n, Complex)) ||
       !(b = NAG_ALLOC(n * 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 LQ factorization of A */
  f08avc(order, m, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08avc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Solve L*Y = B, storing the result in B */
  f07tsc(order, Nag_Lower, Nag_NoTrans, Nag_NonUnitDiag, m,
         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;
    }
  /* Set rows (M+1) to N of B to zero */
  if (m < n)
    {
      for (i = m + 1; i <= n; ++i)
        {
          for (j = 1; j <= nrhs; ++j)
            {
              B(i,j).re = 0.0;
              B(i,j).im = 0.0;
            }
        }
    }

  /* Compute minimum-norm solution X = (Q**H)*B in B */
  f08axc(order, Nag_LeftSide, Nag_ConjTrans, n, nrhs, m, a, pda,
         tau, b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08axc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print minimum-norm solution(s) */
  x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, b, pdb,
         Nag_BracketForm, "%7.4f", "Minimum-norm solution(s)",
         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);
  return exit_status;
}
```

9.2 Program Data

```
f08avc Example Program Data
  3  4  2                                            :Values of M, N and NRHS
 ( 0.28,-0.36) ( 0.50,-0.86) (-0.77,-0.48) ( 1.58, 0.66)
 (-0.50,-1.10) (-1.21, 0.76) (-0.32,-0.24) (-0.27,-1.15)
 ( 0.36,-0.51) (-0.07, 1.33) (-0.75, 0.47) (-0.08, 1.01)   :End of matrix A
 (-1.35, 0.19) ( 4.83,-2.67)
 ( 9.41,-3.56) (-7.28, 3.34)
 (-7.57, 6.93) ( 0.62, 4.53)                               :End of matrix B
```

9.3 Program Results

```
f08avc Example Program Results

 Minimum-norm solution(s)
                     1                  2
 1  (-2.8501, 6.4683)  (-1.1682,-1.8886)
 2  ( 1.6264,-0.7799)  ( 2.8377, 0.7654)
 3  ( 6.9290, 4.6481)  (-1.7610,-0.7041)
 4  ( 1.4048, 3.2400)  ( 1.0518,-1.6365)
```