NAG Library Function Document

nag_zgerqf (f08cvc)

1 Purpose

nag_zgerqf (f08cvc) computes an RQ factorization of a complex m by n matrix A.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_zgerqf (Nag_OrderType order, Integer m, Integer n, Complex a[],
     Integer pda, Complex tau[], NagError *fail)
```

3 Description

nag_zgerqf (f08cvc) forms the RQ factorization of an arbitrary rectangular real m by n matrix. If $m \leq n$, the factorization is given by

$$A = \begin{pmatrix} 0 & R \end{pmatrix} Q,$$

where R is an m by m lower triangular matrix and Q is an n by n unitary matrix. If $m > n$ the factorization is given by

$$A = RQ,$$

where R is an m by n upper trapezoidal matrix and Q is again an n by n unitary matrix. In the case where $m < n$ the factorization can be expressed as

$$A = \begin{pmatrix} 0 & R \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = RQ_2,$$

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

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 9).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

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

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **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 **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

2: **m** – Integer *Input*

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

Constraint: $\mathbf{m} \geq 0$.

3: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 0$.

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;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

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

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the m by n matrix A.

On exit: if $m \leq n$, the upper triangle of the subarray $\mathbf{A}(1 : m, n - m + 1 : n)$ contains the m by m upper triangular matrix R.

If $m \geq n$, the elements on and above the $(m-n)$th subdiagonal contain the m by n upper trapezoidal matrix R; the remaining elements, with the array **tau**, represent the unitary matrix Q as a product of $\min(m, n)$ elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

5: **pda** – Integer *Input*

On entry: the stride separating 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: **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: the scalar factors of the elementary reflectors.

7: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

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 ⟨*value*⟩ had an illegal value.

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*⟩ and **m** = ⟨*value*⟩.
Constraint: **pda** ≥ max(1, **m**).

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pda** ≥ 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 ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_zgerqf (f08cvc) is not threaded by NAG in any implementation.

nag_zgerqf (f08cvc) 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 floating-point operations is approximately $\frac{2}{3}m^2(3n - m)$ if $m \leq n$, or $\frac{2}{3}n^2(3m - n)$ if $m > n$.

To form the unitary matrix Q nag_zgerqf (f08cvc) may be followed by a call to nag_zungrq (f08cwc):

```
nag_zungrq(order, n, n, minmn, a, pda, tau, &fail)
```

where `minmn` = min(m, n), but note that the first dimension of the array **a** must be at least **n**, which may be larger than was required by nag_zgerqf (f08cvc). 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_zungrq(order, m, n, m, a, pda, tau, c, pdc, &fail)
```

To apply Q to an arbitrary real rectangular matrix C, nag_zgerqf (f08cvc) may be followed by a call to nag_zunmrq (f08cxc). For example:

```
nag_zunmrq(Nag_LeftSide, Nag_ConjTrans, n, p, minmn, a, pda, tau, c, pdc,
&fail)
```

forms $C = Q^{\mathrm{H}}C$, where C is n by p.

The real analogue of this function is nag_dgerqf (f08chc).

10 Example

This example finds the minimum norm solution to the underdetermined equations

$$Ax = b$$

where

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

and

$$b = \begin{pmatrix} -1.35 + 0.19i \\ 9.41 - 3.56i \\ -7.57 + 6.93i \end{pmatrix}.$$

The solution is obtained by first obtaining an RQ factorization of the matrix A.

10.1 Program Text

```
/* nag_zgerqf (f08cvc) 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>

int main(void)
{
  /* Scalars */
  Complex       zero = { 0.0, 0.0 };
  Integer       i, j, m, n, nrhs, pda, pdb, pdx;
  Integer       exit_status = 0;
  /* Arrays */
  Complex       *a = 0, *b = 0, *tau = 0, *x = 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]
  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);
```

```
  printf("nag_zgerqf (f08cvc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &m, &n, &nrhs);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &m, &n, &nrhs);
#endif

#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = m;
  pdx = n;
#else
  pda = n;
  pdb = nrhs;
  pdx = 1;
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m*n, Complex)) ||
      !(b = NAG_ALLOC(m*nrhs, Complex)) ||
      !(tau = NAG_ALLOC(m, Complex)) ||
      !(x = NAG_ALLOC(n, Complex)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the matrix A and the vectors 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("%*[^\n]");
#else
  scanf("%*[^\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("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* nag_zgerqf (f08cvc).
   * Compute the RQ factorization of A.
   */
  nag_zgerqf(order, m, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgerqf (f08cvc).\n%s\n", fail.message);
      exit_status = 1;
```

```
      goto END;
    }

  /* nag_zge_copy (f16tfc).
   * Copy the m*nrhs element vector b into x2, where x2 is
   * the vector containing the elements x(n-m+1), ..., x(n) of x.
   */
  nag_zge_copy(order, Nag_NoTrans, m, 1, &B(1, 1), pdb, &x[n - m], pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_ztrtrs (f07tsc).
   * Solve R*y2 = b, storing the result in x2.
   */
  nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, m, 1,
             &A(1, n - m + 1), pda, &x[n - m], pdx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_zload (f16hbc).
   * Set y1 to zero (stored in rows 1 to (n-m) of x).
   */
  nag_zload(n - m, zero, x, 1, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zload (f16hbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_zunmrq (f08cxc).
   * Compute minimum-norm solution x = (Q**H)*y.
   */
  nag_zunmrq(order, Nag_LeftSide, Nag_ConjTrans, n, 1, m, a, pda, tau, x, pdx,
             &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zunmrq (f08cxc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print minimum-norm solution */
  printf("Minimum-norm solution\n");
  for (i = 0; i < n; ++i)
    printf("(%8.4f, %8.4f)\n", x[i].re, x[i].im);

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(tau);
  NAG_FREE(x);

  return exit_status;
}

#undef A
#undef B
```

10.2 Program Data

```
nag_zgerqf (f08cvc) Example Program Data

   3             4             1                         :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)
 ( 9.41,-3.56)
 (-7.57, 6.93)                                            :End of vector b
```

10.3 Program Results

```
nag_zgerqf (f08cvc) Example Program Results

Minimum-norm solution
( -2.8501,   6.4683)
(  1.6264,  -0.7799)
(  6.9290,   4.6481)
(  1.4048,   3.2400)
```