

NAG Library Function Document

nag_zgeqrf (f08asc)

1 Purpose

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

2 Specification

```
#include <nag.h>
#include <nagf08.h>
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 by n matrix. No pivoting is performed.

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

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

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

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

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 9).

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

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: $m \geq 0$.
- 3: **n** – Integer *Input*
On entry: n , the number of columns of the matrix A .
Constraint: $n \geq 0$.
- 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.
The (i, j)th element of the matrix A is stored in
 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ 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 $m \geq n$, 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 row or column elements (depending on the value of **order**) in the array **a**.
Constraints:
if **order** = Nag_ColMajor, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pda} \geq \max(1, \mathbf{n})$.
- 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: further details of the unitary matrix Q .
- 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.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.

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

On entry, $\mathbf{n} = \langle value \rangle$.

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

On entry, $\mathbf{pda} = \langle value \rangle$.

Constraint: $\mathbf{pda} > 0$.

NE_INT_2

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{m} = \langle value \rangle$.

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

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.

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_zgeqrf (f08asc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgeqrf (f08asc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 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 nag_zgeqrf (f08asc) 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 \mathbf{a} must be at least \mathbf{m} , which may be larger than was required by 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:

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

To apply Q to an arbitrary complex rectangular matrix C , `nag_zgeqrf` (f08asc) 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 with column pivoting, use `nag_zgeqpf` (f08bsc).

The real analogue of this function is `nag_dgeqrf` (f08aec).

10 Example

This example solves the linear least squares problems

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

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

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i & -0.05 + 0.41i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i & -0.81 + 0.56i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i & -1.11 + 0.60i \\ -0.37 + 0.38i & 0.19 - 0.54i & -0.98 - 0.36i & 0.22 - 0.20i \\ 0.83 + 0.51i & 0.20 + 0.01i & -0.17 - 0.46i & 1.47 + 1.59i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i & 0.26 + 0.26i \end{pmatrix}$$

and

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

10.1 Program Text

```
/* nag_zgeqrf (f08asc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#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);

    printf("nag_zgeqrf (f08asc) Example Program Results\n");
    /* Skip heading in data file */
    scanf("%*[\n] ");
    scanf("%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)))
    {
        printf("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)
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    }
    scanf("%*[\n] ");
    for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= nrhs; ++j)
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }
    scanf("%*[\n] ");

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

    /* Compute C = (Q**H)*B, storing the result in B */
    /* nag_zunmqr (f08auc).
     * Apply unitary transformation determined by nag_zgeqrf (f08asc)
     * or nag_zgeqpf (f08bsc)
     */
    nag_zunmqr(order, Nag_LeftSide, Nag_ConjTrans, m, nrhs, n, a, pda,
               tau, b, pdb, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_zunmqr (f08auc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

```

```

/* Compute least-squares solution by backsubstitution in R*X = C */
/* nag_ztrtrs (f07tsc).
 * Solution of complex triangular system of linear
 * equations, multiple right-hand sides
 */
nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n, nrhs,
          a, pda, b, pdb, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print least-squares solution(s) */
printf("\n");
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(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, &fail);

if (fail.code != NE_NOERROR)
{
    printf(
        "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(tau);
return exit_status;
}

```

10.2 Program Data

```

nag_zgeqrf (f08asc) Example Program Data
  6  4  2                               :Values of M, N and NRHS
( 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)      :End of matrix A
(-1.54, 0.76) ( 3.17,-2.09)
( 0.12,-1.92) (-6.53, 4.18)
(-9.08,-4.31) ( 7.28, 0.73)
( 7.49, 3.65) ( 0.91,-3.97)
(-5.63,-2.12) (-5.46,-1.64)
( 2.37, 8.03) (-2.84,-5.86)                                  :End of matrix B

```

10.3 Program Results

```

nag_zgeqrf (f08asc) Example Program Results

Least-squares solution(s)
      1      2
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)

```
