

# NAG Library Function Document

## nag\_zgeqrt (f08apc)

### 1 Purpose

nag\_zgeqrt (f08apc) recursively computes, with explicit blocking, the  $QR$  factorization of a complex  $m$  by  $n$  matrix.

### 2 Specification

```
#include <nag.h>
#include <nagf08.h>

void nag_zgeqrt (Nag_OrderType order, Integer m, Integer n, Integer nb,
                Complex a[], Integer pda, Complex t[], Integer pdt, NagError *fail)
```

### 3 Description

nag\_zgeqrt (f08apc) forms the  $QR$  factorization of an arbitrary rectangular complex  $m$  by  $n$  matrix. No pivoting is performed.

It differs from nag\_zgeqrf (f08asc) in that it: requires an explicit block size; stores reflector factors that are upper triangular matrices of the chosen block size (rather than scalars); and recursively computes the  $QR$  factorization based on the algorithm of Elmroth and Gustavson (2000).

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 upper 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 represents a  $QR$  factorization of the first  $k$  columns of the original matrix  $A$ .

## 4 References

Elmroth E and Gustavson F (2000) Applying Recursion to Serial and Parallel *QR* Factorization Leads to Better Performance *IBM Journal of Research and Development*. (Volume 44) 4 605–624

Golub G H and Van Loan C F (2012) *Matrix Computations* (4th 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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** ≥ 0.

3: **n** – Integer *Input*

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

*Constraint:* **n** ≥ 0.

4: **nb** – Integer *Input*

*On entry:* the explicitly chosen block size to be used in computing the *QR* factorization. See Section 9 for details.

*Constraints:*

**nb** ≥ 1;  
if min(**m**, **n**) > 0, **nb** ≤ min(**m**, **n**).

5: **a**[*dim*] – Complex *Input/Output*

**Note:** the dimension, *dim*, of the array **a** must be at least

max(1, **pda** × **n**) when **order** = Nag\_ColMajor;  
max(1, **m** × **pda**) when **order** = Nag\_RowMajor.

The (*i*, *j*)th element of the matrix *A* is stored in

**a**[(*j* – 1) × **pda** + *i* – 1] when **order** = Nag\_ColMajor;  
**a**[(*i* – 1) × **pda** + *j* – 1] when **order** = Nag\_RowMajor.

*On entry:* the *m* by *n* matrix *A*.

*On exit:* if *m* ≥ *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.

- 6: **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, **m**);  
 if **order** = Nag\_RowMajor, **pda**  $\geq$  max(1, **n**).
- 7: **t[*dim*]** – Complex *Output*  
**Note:** the dimension, *dim*, of the array **t** must be at least  
 max(1, **pdt**  $\times$  min(**m**, **n**)) when **order** = Nag\_ColMajor;  
 max(1, **nb**  $\times$  **pdt**) when **order** = Nag\_RowMajor.  
 The (*i*, *j*)th element of the matrix *T* is stored in  
**t**[(*j* – 1)  $\times$  **pdt** + *i* – 1] when **order** = Nag\_ColMajor;  
**t**[(*i* – 1)  $\times$  **pdt** + *j* – 1] when **order** = Nag\_RowMajor.  
*On exit:* further details of the unitary matrix *Q*. The number of blocks is  $b = \lceil \frac{k}{\mathbf{nb}} \rceil$ , where  $k = \min(m, n)$  and each block is of order **nb** except for the last block, which is of order  $k - (b - 1) \times \mathbf{nb}$ . For each of the blocks, an upper triangular block reflector factor is computed:  $T_1, T_2, \dots, T_b$ . These are stored in the **nb** by *n* matrix *T* as  $T = [T_1 | T_2 | \dots | T_b]$ .
- 8: **pdt** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **t**.  
*Constraints:*  
 if **order** = Nag\_ColMajor, **pdt**  $\geq$  **nb**;  
 if **order** = Nag\_RowMajor, **pdt**  $\geq$  max(1, min(**m**, **n**)).
- 9: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument *<value>* had an illegal value.

### NE\_INT

On entry, **m** = *<value>*.

Constraint: **m**  $\geq$  0.

On entry, **n** = *<value>*.

Constraint: **n**  $\geq$  0.

### NE\_INT\_2

On entry, **pda** = *<value>* and **m** = *<value>*.

Constraint: **pda**  $\geq$  max(1, **m**).

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

On entry, **pdt** =  $\langle value \rangle$  and **nb** =  $\langle value \rangle$ .

Constraint: **pdt**  $\geq \mathbf{nb}$ .

### NE\_INT\_3

On entry, **nb** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **nb**  $\geq 1$  and

if  $\min(\mathbf{m}, \mathbf{n}) > 0$ , **nb**  $\leq \min(\mathbf{m}, \mathbf{n})$ .

On entry, **pdt** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **pdt**  $\geq \max(1, \min(\mathbf{m}, \mathbf{n}))$ .

### 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.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation 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  $\epsilon$  is the *machine precision*.

## 8 Parallelism and Performance

nag\_zgeqrt (f08apc) 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 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 apply  $Q$  to an arbitrary complex rectangular matrix  $C$ , nag\_zgeqrt (f08apc) may be followed by a call to nag\_zgemqrt (f08aqc). For example,

```
nag_zgemqrt(order, Nag_LeftSide, Nag_ConjTrans, m, p, MIN(m, n), nb, a, pda,
            t, pdt, c, pdc, &fail)
```

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

To form the unitary matrix  $Q$  explicitly, simply initialize the  $m$  by  $m$  matrix  $C$  to the identity matrix and form  $C = QC$  using nag\_zgemqrt (f08aqc) as above.

The block size, **nb**, used by nag\_zgeqrt (f08apc) is supplied explicitly through the interface. For moderate and large sizes of matrix, the block size can have a marked effect on the efficiency of the algorithm with the optimal value being dependent on problem size and platform. A value of

$\mathbf{nb} = 64 \ll \min(m, n)$  is likely to achieve good efficiency and it is unlikely that an optimal value would exceed 340.

To compute a  $QR$  factorization with column pivoting, use `nag_ztpqrt` (f08bpc) or `nag_zgeqpf` (f08bsc). The real analogue of this function is `nag_dgeqrt` (f08abc).

## 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

$$B = \begin{pmatrix} -2.09 + 1.93i & 3.26 - 2.70i \\ 3.34 - 3.53i & -6.22 + 1.16i \\ -4.94 - 2.04i & 7.94 - 3.13i \\ 0.17 + 4.23i & 1.04 - 4.26i \\ -5.19 + 3.63i & -2.31 - 2.12i \\ 0.98 + 2.53i & -1.39 - 4.05i \end{pmatrix}.$$

### 10.1 Program Text

```

/* nag_zgeqrt (f08apc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
    /* Scalars */
    double rnorm;
    Integer exit_status = 0;
    Integer pda, pdb, pdt;
    Integer i, j, m, n, nb, nrhs;
    /* Arrays */
    Complex *a = 0, *b = 0, *t = 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_zgeqrt (f08apc) Example Program Results\n\n");
    fflush(stdout);

    /* 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
    nb = MIN(m, n);
    if (!(a = NAG_ALLOC(m * n, Complex)) ||
        !(b = NAG_ALLOC(m * nrhs, Complex)) ||
        !(t = NAG_ALLOC(nb * MIN(m, n), Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = m;
    pdt = nb;
#else
    pda = n;
    pdb = nrhs;
    pdt = MIN(m, n);
#endif

    /* Read A and 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_zgeqrt (f08apc).
     * Compute the QR factorization of A by recursive algorithm.
     */
    nag_zgeqrt(order, m, n, nb, a, pda, t, pdt, &fail);
    if (fail.code != NE_NOERROR) {

```

```

    printf("Error from nag_zgeqrt (f08apc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_zgemqrt (f08aqc).
 * Compute C = (C1) = (Q`H)*B, storing the result in B.
 *          (C2)
 * by applying Q`H from left.
 */
nag_zgemqrt(order, Nag_LeftSide, Nag_ConjTrans, m, nrhs, n, nb, a, pda, t,
            pdt, b, pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgemqrt (f08aqc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
}

/* nag_ztrtrs (f07tsc).
 * Compute least squares solutions by back-substitution in R*X = C1.
 */
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 = 3;
    goto END;
}

/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print least squares solutions.
 */
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 = 4;
    goto END;
}

printf("\n Square root(s) of the residual sum(s) of squares\n");
for (j = 1; j <= nrhs; j++) {
    /* nag_zge_norm (f16uac).
     * Compute and print estimate of the square root of the residual
     * sum of squares.
     */
    nag_zge_norm(order, Nag_FrobeniusNorm, m - n, 1, &B(n + 1, j), pdb,
                 &rnorm, &fail);
    if (fail.code != NE_NOERROR) {
        printf("\nError from nag_zge_norm (f16uac).\n%s\n", fail.message);
        exit_status = 5;
        goto END;
    }
    printf("  %11.2e ", rnorm);
}
printf("\n");

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(t);

return exit_status;
}

```

## 10.2 Program Data

nag\_zgeqrt (f08apc) Example Program Data

```

      6              4              2              : 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) : matrix A

(-2.09, 1.93) ( 3.26,-2.70)
( 3.34,-3.53) (-6.22, 1.16)
(-4.94,-2.04) ( 7.94,-3.13)
( 0.17, 4.23) ( 1.04,-4.26)
(-5.19, 3.63) (-2.31,-2.12)
( 0.98, 2.53) (-1.39,-4.05) : matrix B

```

## 10.3 Program Results

nag\_zgeqrt (f08apc) Example Program Results

Least squares solution(s)

```

              1              2
1 (-0.5044,-1.2179) ( 0.7629, 1.4529)
2 (-2.4281, 2.8574) ( 5.1570,-3.6089)
3 ( 1.4872,-2.1955) (-2.6518, 2.1203)
4 ( 0.4537, 2.6904) (-2.7606, 0.3318)

```

Square root(s) of the residual sum(s) of squares  
 6.88e-02      1.87e-01

---