# NAG Library Function Document

# nag_zggqrf (f08zsc)

## 1 Purpose

nag_zggqrf (f08zsc) computes a generalized $QR$ factorization of a complex matrix pair $(A,B)$, where $A$ is an $n$ by $m$ matrix and $B$ is an $n$ by $p$ matrix.

## 2 Specification

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

```
void nag_zggqrf (Nag_OrderType order, Integer n, Integer m, Integer p,
     Complex a[], Integer pda, Complex taua[], Complex b[], Integer pdb,
     Complex taub[], NagError *fail)
```

## 3 Description

nag_zggqrf (f08zsc) forms the generalized $QR$ factorization of an $n$ by $m$ matrix $A$ and an $n$ by $p$ matrix $B$

$$A = QR, \quad B = QTZ,$$

where $Q$ is an $n$ by $n$ unitary matrix, $Z$ is a $p$ by $p$ unitary matrix and $R$ and $T$ are of the form

$$R = \begin{cases} \begin{array}{c} \\ m \\ n-m \end{array} \!\!\begin{array}{c} m \\ \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array}, & \text{if } n \geq m; \\ \begin{array}{c} \\ n \end{array} \!\!\begin{array}{cc} n & m-n \\ \multicolumn{2}{c}{\begin{pmatrix} R_{11} & R_{12} \end{pmatrix}} \end{array}, & \text{if } n < m, \end{cases}$$

with $R_{11}$ upper triangular,

$$T = \begin{cases} \begin{array}{c} \\ n \end{array} \!\!\begin{array}{cc} p-n & n \\ \multicolumn{2}{c}{\begin{pmatrix} 0 & T_{12} \end{pmatrix}} \end{array}, & \text{if } n \leq p, \\ \begin{array}{c} \\ n-p \\ p \end{array} \!\!\begin{array}{c} p \\ \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \end{array}, & \text{if } n > p, \end{cases}$$

with $T_{12}$ or $T_{21}$ upper triangular.

In particular, if $B$ is square and nonsingular, the generalized $QR$ factorization of $A$ and $B$ implicitly gives the $QR$ factorization of $B^{-1}A$ as

$$B^{-1}A = Z^{\mathrm{H}}\left(T^{-1}R\right).$$

## 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

Anderson E, Bai Z and Dongarra J (1992) Generalized *QR* factorization and its applications *Linear Algebra Appl. (Volume 162–164)* 243–271

Hammarling S (1987) The numerical solution of the general Gauss-Markov linear model *Mathematics in Signal Processing* (eds T S Durrani, J B Abbiss, J E Hudson, R N Madan, J G McWhirter and T A Moore) 441–456 Oxford University Press

Paige C C (1990) Some aspects of generalized $QR$ factorizations . *In Reliable Numerical Computation* (eds M G Cox and S Hammarling) 73–91 Oxford University Press

## 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: **n** – Integer *Input*

*On entry*: $n$, the number of rows of the matrices $A$ and $B$.

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

3: **m** – Integer *Input*

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

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

4: **p** – Integer *Input*

*On entry*: $p$, the number of columns of the matrix $B$.

*Constraint*: $\mathbf{p} \geq 0$.

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

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

$\max(1, \mathbf{pda} \times \mathbf{m})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pda})$ 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 $n$ by $m$ matrix $A$.

*On exit*: the elements on and above the diagonal of the array contain the $\min(n, m)$ by $m$ upper trapezoidal matrix $R$ ($R$ is upper triangular if $n \geq m$); the elements below the diagonal, with the array **taua**, represent the unitary matrix $Q$ as a product of $\min(n, m)$ elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

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

7: **taua**[**min**(**n**, **m**)] – Complex *Output*

*On exit*: the scalar factors of the elementary reflectors which represent the unitary matrix $Q$.

8: **b**[$dim$] – Complex *Input/Output*

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

max(1, **pdb** × **p**) when **order** = Nag_ColMajor;
max(1, **n** × **pdb**) when **order** = Nag_RowMajor.

Where **B**($i, j$) appears in this document, it refers to the array element

**b**[($j - 1$) × **pdb** + $i - 1$] when **order** = Nag_ColMajor;
**b**[($i - 1$) × **pdb** + $j - 1$] when **order** = Nag_RowMajor.

*On entry*: the $n$ by $p$ matrix $B$.

*On exit*: if $n \leq p$, the upper triangle of the subarray **B**($1 : n, p - n + 1 : p$) contains the $n$ by $n$ upper triangular matrix $T_{12}$.

If $n > p$, the elements on and above the $(n - p)$th subdiagonal contain the $n$ by $p$ upper trapezoidal matrix $T$; the remaining elements, with the array **taub**, represent the unitary matrix $Z$ as a product of elementary reflectors (see Section 3.3.6 in the f08 Chapter Introduction).

9: **pdb** – Integer *Input*

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

*Constraints*:

if **order** = Nag_ColMajor, **pdb** ≥ max(1, **n**);
if **order** = Nag_RowMajor, **pdb** ≥ max(1, **p**).

10: **taub**[**min**(**n**, **p**)] – Complex *Output*

*On exit*: the scalar factors of the elementary reflectors which represent the unitary matrix $Z$.

11: **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** ≥ 0.

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

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

On entry, **pda** $= \langle value \rangle$.
Constraint: **pda** $> 0$.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** $> 0$.

**NE_INT_2**

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

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

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

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

**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 generalized $QR$ factorization is the exact factorization for nearby matrices $(A + E)$ and $(B + F)$, where

$$\|E\|_2 = O\epsilon\|A\|_2 \quad \text{and} \quad \|F\|_2 = O\epsilon\|B\|_2,$$

and $\epsilon$ is the ***machine precision***.

# 8 Parallelism and Performance

nag_zggqrf (f08zsc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zggqrf (f08zsc) 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 unitary matrices $Q$ and $Z$ may be formed explicitly by calls to nag_zungqr (f08atc) and nag_zungrq (f08cwc) respectively. nag_zunmqr (f08auc) may be used to multiply $Q$ by another matrix and nag_zunmrq (f08cxc) may be used to multiply $Z$ by another matrix.

The real analogue of this function is nag_dggqrf (f08zec).

## 10 Example

This example solves the general Gauss–Markov linear model problem

$$\min_x \|y\|_2 \quad \text{subject to} \quad d = Ax + By$$

where

$$A = \begin{pmatrix} 0.96 - 0.81i & -0.03 + 0.96i & -0.91 + 2.06i \\ -0.98 + 1.98i & -1.20 + 0.19i & -0.66 + 0.42i \\ 0.62 - 0.46i & 1.01 + 0.02i & 0.63 - 0.17i \\ 1.08 - 0.28i & 0.20 - 0.12i & -0.07 + 1.23i \end{pmatrix},$$

$$B = \begin{pmatrix} 0.5 - 1.0i & 0 & 0 & 0 \\ 0 & 1.0 - 2.0i & 0 & 0 \\ 0 & 0 & 2.0 - 3.0i & 0 \\ 0 & 0 & 0 & 5.0 - 4.0i \end{pmatrix}$$

and

$$d = \begin{pmatrix} 6.00 - 0.40i \\ -5.27 + 0.90i \\ 2.72 - 2.13i \\ -1.30 - 2.80i \end{pmatrix}.$$

The solution is obtained by first computing a generalized $QR$ factorization of the matrix pair $(A, B)$. The example illustrates the general solution process, although the above data corresponds to a simple weighted least squares problem.

### 10.1 Program Text

```
/* nag_zggqrf (f08zsc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>

int main(void)
{
  /* Scalars */
  Complex alpha, beta;
  Complex zero = { 0.0, 0.0 };
  double rnorm;
  Integer i, j, m, n, nm, p, pda, pdb, pdd, pnm, zrow;
  Integer exit_status = 0;

  /* Arrays */
  Complex *a = 0, *b = 0, *d = 0, *taua = 0, *taub = 0, *y = 0;

  /* Nag Types */
  NagError fail;
  Nag_OrderType order;

#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_zggqrf (f08zsc) 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]", &n, &m, &p);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &n, &m, &p);
#endif
  if (n < 0 || m < 0 || p < 0) {
    printf("Invalid n, m or p\n");
    exit_status = 1;
    goto END;
  }

#ifdef NAG_COLUMN_MAJOR
  pda = n;
  pdb = n;
  pdd = n;
#else
  pda = m;
  pdb = p;
  pdd = 1;
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * m, Complex)) ||
      !(b = NAG_ALLOC(n * p, Complex)) ||
      !(d = NAG_ALLOC(n, Complex)) ||
      !(taua = NAG_ALLOC(MIN(n, m), Complex)) ||
      !(taub = NAG_ALLOC(MIN(n, p), Complex)) || !(y = NAG_ALLOC(p, Complex)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read A, B and d from data file */
  for (i = 1; i <= n; ++i)
    for (j = 1; j <= m; ++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 <= n; ++i)
    for (j = 1; j <= p; ++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
#ifdef _WIN32
  for (i = 0; i < n; ++i)
    scanf_s(" ( %lf , %lf )", &d[i].re, &d[i].im);
#else
  for (i = 0; i < n; ++i)
    scanf(" ( %lf , %lf )", &d[i].re, &d[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Compute the generalized QR factorization of (A,B) as
   * A = Q*(R),    B = Q*(T11 T12)*Z
   *       (0)          ( 0  T22)
   * using  nag_dggqrf (f08zec).
   */
  nag_zggqrf(order, n, m, p, a, pda, taua, b, pdb, taub, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zggqrf (f08zsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Solve weighted least squares problem for case n > m */
  if (n <= m)
    goto END;

  nm = n - m;
  pnm = p - nm;
  /* Multiply Q^H through d = Ax + By to get
   *     (c1) = Q^H * d = (R) * x + (T11 T12) * Z * (y1)
   *     (c2)             (0)       ( 0  T22)       (y2)
   * Compute C using nag_zunmqr (f08auc).
   */
  nag_zunmqr(order, Nag_LeftSide, Nag_ConjTrans, n, 1, m, a, pda, taua, d,
             pdd, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zunmqr (f08auc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Let Z*(y1) = (w1) and solving for w2 we have to solve the triangular sytem
   *       (y2) = (w2)
   *                      T22 * w2 = c2
   * This is done by putting c2 in y2 and backsolving to get w2 in y2.
   *
   * Copy c2 (at d[m]) into y2 using nag_zge_copy (f16tfc).
   */
  nag_zge_copy(Nag_ColMajor, Nag_NoTrans, nm, 1, &d[m], n - m, &y[pnm], nm,
               &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Solve T22*w2 = c2 using nag_ztrtrs (f07tsc).
   * T22 is stored in a submatrix of matrix B of dimension n-m by n-m
   * with first element at B(m+1,p-(n-m)+1). y2 is stored from y[p-(n-m)].
   */
  nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, nm, 1,
             &B(m + 1, pnm + 1), pdb, &y[pnm], nm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
```

```
}

/* set w1 = 0 for minimum norm y. */
nag_zload(pnm, zero, y, 1, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_zload (f16hbc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Compute estimate of the square root of the residual sum of squares
 * norm(y) = norm(w2) with y1 = 0 using  nag_dge_norm (f16uac).
 */
nag_zge_norm(Nag_ColMajor, Nag_FrobeniusNorm, nm, 1, &y[pnm], nm, &rnorm,
             &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* The top half of the system remains:
 *     (c1) = Q^H * d = (R) * x + (T11 T12) * ( 0)
 *                                            (w2)
 * =>      c1 = R * x + T12 * w2
 * =>   R * x = c1    - T12 * w2;
 *
 * first form d = c1 - T12*w2 where c1 is stored in d
 * using nag_zgemv (f16sac).
 */
alpha = nag_complex(-1.0, 0.0);
beta = nag_complex(1.0, 0.0);
nag_zgemv(order, Nag_NoTrans, m, nm, alpha, &B(1, pnm + 1), pdb, &y[pnm], 1,
          beta, d, 1, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_zgemv (f16sac).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Next, solve R * x = d for x (in d) where R is stored in leading submatrix
 * of A in a. This gives the least squares solution x in d.
 * Using nag_dtrtrs (f07tec).
 */
nag_ztrtrs(order, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, m, 1, a, pda, d,
           pdd, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_ztrtrs (f07tsc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Compute the minimum norm residual vector y = (Z^T)*w
 * using nag_dzunmrq (f08cxc).
 */
zrow = MAX(1, n - p + 1);
nag_zunmrq(order, Nag_LeftSide, Nag_ConjTrans, p, 1, MIN(n, p), &B(zrow, 1),
           pdb, taub, y, pdd, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_zunmrq (f08cxc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print least squares solution x */
printf("Generalized least squares solution\n");
for (i = 0; i < m; ++i)
  printf(" (%9.4f, %9.4f)%s", d[i].re, d[i].im, i % 3 == 2 ? "\n" : "");

/* Print residual vector y */
printf("\n");
```

```
  printf("\nResidual vector\n");
  for (i = 0; i < p; ++i)
    printf(" (%9.2e, %9.2e)%s", y[i].re, y[i].im, i % 3 == 2 ? "\n" : "");

  /* Print estimate of the square root of the residual sum of  squares. */
  printf("\n\nSquare root of the residual sum of squares\n");
  printf("%11.2e\n", rnorm);

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(d);
  NAG_FREE(taua);
  NAG_FREE(taub);
  NAG_FREE(y);

  return exit_status;
}
```

## 10.2 Program Data

```
nag_zggqrf (f08zsc) Example Program Data

  4            3            4                         : n, m and p

( 0.96,-0.81) (-0.03, 0.96) (-0.91, 2.06)
(-0.98, 1.98) (-1.20, 0.19) (-0.66, 0.42)
( 0.62,-0.46) ( 1.01, 0.02) ( 0.63,-0.17)
( 1.08,-0.28) ( 0.20,-0.12) (-0.07, 1.23)             : matrix A

( 0.50,-1.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 1.00,-2.00) ( 0.00, 0.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 0.00, 0.00) ( 2.00,-3.00) ( 0.00, 0.00)
( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 5.00,-4.00) : matrix B

( 6.00,-0.40)
(-5.27, 0.90)
( 2.72,-2.13)
(-1.30,-2.80)                                         : vector d
```

## 10.3 Program Results

```
nag_zggqrf (f08zsc) Example Program Results

Generalized least squares solution
 (  -0.9846,    1.9950) (   3.9929,   -4.9748) (  -3.0026,    0.9994)


Residual vector
 ( 1.26e-04, -4.66e-04) ( 1.11e-03, -8.61e-04) ( 3.84e-03, -1.82e-03)
 ( 2.03e-03,  3.02e-03)

Square root of the residual sum of squares
   5.79e-03
```