

# NAG Library Function Document

## nag\_dtzrzf (f08bhc)

### 1 Purpose

nag\_dtzrzf (f08bhc) reduces the  $m$  by  $n$  ( $m \leq n$ ) real upper trapezoidal matrix  $A$  to upper triangular form by means of orthogonal transformations.

### 2 Specification

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

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

### 3 Description

The  $m$  by  $n$  ( $m \leq n$ ) real upper trapezoidal matrix  $A$  given by

$$A = \begin{pmatrix} R_1 & R_2 \end{pmatrix},$$

where  $R_1$  is an  $m$  by  $m$  upper triangular matrix and  $R_2$  is an  $m$  by  $(n - m)$  matrix, is factorized as

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

where  $R$  is also an  $m$  by  $m$  upper triangular matrix and  $Z$  is an  $n$  by  $n$  orthogonal matrix.

### 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>

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

3: **n** – Integer *Input*

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

*Constraint:*  $n \geq 0$ .

4: **a**[*dim*] – double *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

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

*On entry:* the leading *m* by *n* upper trapezoidal part of the array **a** must contain the matrix to be factorized.

*On exit:* the leading *m* by *m* upper triangular part of **a** contains the upper triangular matrix *R*, and elements **m** + 1 to **n** of the first *m* rows of **a**, with the array **tau**, represent the orthogonal matrix *Z* as a product of *m* 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** ≥ max(1, **m**);  
 if **order** = Nag\_RowMajor, **pda** ≥ max(1, **n**).

6: **tau**[*dim*] – double *Output*

**Note:** the dimension, *dim*, of the array **tau** must be at least max(1, **m**).

*On exit:* the scalar factors of the elementary reflectors.

7: **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 3.2.1.2 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, **pda** = *⟨value⟩*.

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

**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 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 3.6.5 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\_dtzrzf (f08bhc) 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  $4m^2(n - m)$ .

The complex analogue of this function is nag\_ztzrzf (f08bvc).

**10 Example**

This example solves the linear least squares problems

$$\min_x \|b_j - Ax_j\|_2, \quad j = 1, 2$$

for the minimum norm solutions  $x_1$  and  $x_2$ , where  $b_j$  is the  $j$ th column of the matrix  $B$ ,

$$A = \begin{pmatrix} -0.09 & 0.14 & -0.46 & 0.68 & 1.29 \\ -1.56 & 0.20 & 0.29 & 1.09 & 0.51 \\ -1.48 & -0.43 & 0.89 & -0.71 & -0.96 \\ -1.09 & 0.84 & 0.77 & 2.11 & -1.27 \\ 0.08 & 0.55 & -1.13 & 0.14 & 1.74 \\ -1.59 & -0.72 & 1.06 & 1.24 & 0.34 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 7.4 & 2.7 \\ 4.2 & -3.0 \\ -8.3 & -9.6 \\ 1.8 & 1.1 \\ 8.6 & 4.0 \\ 2.1 & -5.7 \end{pmatrix}.$$

The solution is obtained by first obtaining a  $QR$  factorization with column pivoting of the matrix  $A$ , and then the  $RZ$  factorization of the leading  $k$  by  $k$  part of  $R$  is computed, where  $k$  is the estimated rank of  $A$ . A tolerance of 0.01 is used to estimate the rank of  $A$  from the upper triangular factor,  $R$ .

## 10.1 Program Text

```

/* nag_dtzrzf (f08bhc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double d, f, tol;
    Integer i, j, k, m, n, nrhs, pda, pdb;
    Integer exit_status = 0;
    /* Arrays */
    double *a = 0, *b = 0, *rnorm = 0, *tau = 0, *work = 0;
    Integer *jpvt = 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_dtzrzf (f08bhc) 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;
#else
    pda = n;
    pdb = nrhs;
#endif

    /* Allocate memory */
    if (!(a = NAG_ALLOC(m * n, double)) ||
        !(b = NAG_ALLOC(m * nrhs, double)) ||
        !(rnorm = NAG_ALLOC(nrhs, double)) ||
        !(tau = NAG_ALLOC(n, double)) ||

```

```

        !(work = NAG_ALLOC(n, double)) || !(jpvt = NAG_ALLOC(n, Integer)))
    {
        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)
#ifdef _WIN32
        scanf_s("%lf", &A(i, j));
#else
        scanf("%lf", &A(i, j));
#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", &B(i, j));
#else
            scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
            scanf_s("%*[\n]");
#else
            scanf("%*[\n]");
#endif

    /* nag_iloadd (f16dbc).
     * Initialize jpvt to be zero so that all columns are free.
     */
    nag_iloadd(n, 0, jpvt, 1, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_iloadd (f16dbc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* nag_dgeqp3 (f08bfc).
     * Compute the QR factorization of A with column pivoting as
     *  $A = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} (P^T)$ 
     */
    nag_dgeqp3(order, m, n, a, pda, jpvt, tau, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dgeqp3 (f08bfc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* nag_dormqr (f08ckc).
     * Compute  $C = (C1) = (Q^T) * B$ , storing the result in b.
     * (C2)
     */
    nag_dormqr(order, Nag_LeftSide, Nag_Trans, m, nrhs, n, a, pda, tau, b, pdb,
                &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dormqr (f08ckc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Choose tol to reflect the relative accuracy of the input data */
    tol = 0.01;

```

```

/* Determine and print the rank, k, of R relative to tol */
for (k = 1; k <= n; ++k)
    if ((f = A(k, k), fabs(f)) <= tol * (d = A(1, 1), fabs(d)))
        break;
--k;

printf("Tolerance used to estimate the rank of A\n");
printf("%11.2e\n", tol);
printf("Estimated rank of A\n");
printf("%8" NAG_IFMT "\n\n", k);

/* nag_dtzrzf (f08bhc).
 * Compute the RZ factorization of the k by k part of R as
 * (R11 R12) = (T 0)*Z
 */
nag_dtzrzf(order, k, n, a, pda, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtzrzf (f08bhc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dtrsm (f16yjc).
 * Compute least squares solutions of triangular problems by
 * back substitution in T*Y1 = C1, storing the result in b.
 */
nag_dtrsm(order, Nag_LeftSide, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag,
          k, nrhs, 1.0, a, pda, b, pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrsm (f16yjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dge_norm (f16rac).
 * Compute estimates of the square roots of the residual sums of
 * squares (2-norm of each of the columns of C2).
 */
for (j = 1; j <= nrhs; ++j) {
    nag_dge_norm(order, Nag_FrobeniusNorm, m - k, 1, &B(k + 1, j), pdb,
                &rnorm[j - 1], &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}

/* nag_dge_load (f16qhc).
 * Set the remaining elements of the solutions to zero (to give
 * the minimum-norm solutions), Y2 = 0.
 */
nag_dge_load(order, n - k, nrhs, 0.0, 0.0, &B(k + 1, 1), pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dge_load (f16qhc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dormrz (f08bkc).
 * Form W = (Z^T)*Y.
 */
nag_dormrz(order, Nag_LeftSide, Nag_Trans, n, nrhs, k, n - k, a, pda, tau,
          b, pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dormrz (f08bkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Permute the least squares solutions stored in B to give X = P*W */

```

```

for (j = 1; j <= nrhs; ++j) {
  for (i = 1; i <= n; ++i)
    work[jpvt[i - 1] - 1] = B(i, j);
  for (i = 1; i <= n; ++i)
    B(i, j) = work[i - 1];
}

/* nag_gen_real_mat_print (x04cac).
 * Print least squares solutions.
 */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs,
                       b, pdb, "Least squares solution(s)", 0, &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print the square roots of the residual sums of squares */
printf("\nSquare root(s) of the residual sum(s) of squares\n");

for (j = 0; j < nrhs; ++j)
  printf("%11.2e%s", rnorm[j], j % 6 == 5 ? "\n" : " ");

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(rnorm);
  NAG_FREE(tau);
  NAG_FREE(work);
  NAG_FREE(jpvt);
  return exit_status;
}

#undef A
#undef B

```

## 10.2 Program Data

nag\_dtzrzf (f08bhc) Example Program Data

```

  6  5  2                               :Values of m, n and nrhs

-0.09  0.14 -0.46  0.68  1.29
-1.56  0.20  0.29  1.09  0.51
-1.48 -0.43  0.89 -0.71 -0.96
-1.09  0.84  0.77  2.11 -1.27
 0.08  0.55 -1.13  0.14  1.74
-1.59 -0.72  1.06  1.24  0.34 :End of matrix A

 7.4   2.7
 4.2  -3.0
-8.3  -9.6
 1.8   1.1
 8.6   4.0
 2.1  -5.7                               :End of matrix B

```

## 10.3 Program Results

nag\_dtzrzf (f08bhc) Example Program Results

```

Tolerance used to estimate the rank of A
 1.00e-02
Estimated rank of A
 4

```

```

Least squares solution(s)
      1      2
1      0.6344      3.6258

```

2	0.9699	1.8284
3	-1.4402	-1.6416
4	3.3678	2.4307
5	3.3992	0.2818

Square root(s) of the residual sum(s) of squares  
2.54e-02 3.65e-02

---