NAG Library Function Document

nag_ztzrzf (f08bvc)

1 Purpose

nag_ztzrzf (f08bvc) reduces the m by n ($m \leq n$) complex upper trapezoidal matrix A to upper triangular form by means of unitary transformations.

2 Specification

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

3 Description

The m by n ($m \leq n$) complex 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 unitary 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** ≥ 0.

3: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

4: **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[$(j-1) \times$ **pda** $+ i - 1$] when **order** = Nag_ColMajor;
a[$(i-1) \times$ **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 unitary 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] – Complex *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** = ⟨*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 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 ϵ is the ***machine precision***.

8 Parallelism and Performance

nag_ztzrzf (f08bvc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The total number of floating-point operations is approximately $16m^2(n-m)$.

The real analogue of this function is nag_dtzrzf (f08bhc).

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 jth column of the matrix B,

$$A = \begin{pmatrix} 0.47 - 0.34i & -0.40 + 0.54i & 0.60 + 0.01i & 0.80 - 1.02i \\ -0.32 - 0.23i & -0.05 + 0.20i & -0.26 - 0.44i & -0.43 + 0.17i \\ 0.35 - 0.60i & -0.52 - 0.34i & 0.87 - 0.11i & -0.34 - 0.09i \\ 0.89 + 0.71i & -0.45 - 0.45i & -0.02 - 0.57i & 1.14 - 0.78i \\ -0.19 + 0.06i & 0.11 - 0.85i & 1.44 + 0.80i & 0.07 + 1.14i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -1.08 - 2.59i & 2.22 + 2.35i \\ -2.61 - 1.49i & 1.62 - 1.48i \\ 3.13 - 3.61i & 1.65 + 3.43i \\ 7.33 - 8.01i & -0.98 + 3.08i \\ 9.12 + 7.63i & -2.84 + 2.78i \end{pmatrix}.$$

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_ztzrzf (f08bvc) 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 <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Complex one = { 1.0, 0.0 };
  Complex zero = { 0.0, 0.0 };
  double tol;
  Integer i, j, k, m, n, nrhs, pda, pdb, pdw;
  Integer exit_status = 0;
  /* Arrays */
  Complex *a = 0, *b = 0, *tau = 0, *work = 0;
  double *rnorm = 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_ztzrzf (f08bvc) 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;
  pdw = m;
```

```
#else
  pda = n;
  pdb = nrhs;
  pdw = 1;
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * n, Complex)) ||
      !(b = NAG_ALLOC(m * nrhs, Complex)) ||
      !(tau = NAG_ALLOC(n, Complex)) ||
      !(work = NAG_ALLOC(n, Complex)) ||
      !(rnorm = NAG_ALLOC(nrhs, 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 , %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_iload (f16dbc).
   * Initialize jpvt to be zero so that all columns are free.
   */
  nag_iload(n, 0, jpvt, 1, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_iload (f16dbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_zgeqp3 (f08btc).
   * Compute the QR factorization of A with column pivoting as
   * A = Q*(R11 R12)*(P^T)
   *       ( 0  R22)
   */
  nag_zgeqp3(order, m, n, a, pda, jpvt, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zgeqp3 (f08btc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_zunmqr (f08auc).
   * Compute C = (C1) = (Q^H)*B, storing the result in b.
   *             (C2)
   */
```

```
  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;
  }

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

  /* nag_complex_abs (a02dbc).
   * Determine and print the rank, k, of R relative to tol.
   */
  for (k = 1; k <= n; ++k)
    if (nag_complex_abs(A(k, k)) <= tol * nag_complex_abs(A(1, 1)))
      break;
  --k;

  printf("Tolerance used to estimate the rank of A\n");
  printf("%11.2e\n", tol);
  printf("Estimated rank of A\n");
  printf("%6" NAG_IFMT "\n\n", k);

  /* nag_ztzrzf (f08bvc).
   * Compute the RZ factorization of the k by k part of R as
   * (R1 R2) = (T 0)*Z.
   */
  nag_ztzrzf(order, k, n, a, pda, tau, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztzrzf (f08bvc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_ztrsm (f16zjc).
   * Compute least squares solutions of triangular problems by
   * back substitution in T*Y1 = C1, storing the result in b.
   */
  nag_ztrsm(order, Nag_LeftSide, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, k,
            nrhs, one, a, pda, b, pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ztrsm (f16zjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_zge_norm (f16uac).
   * 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_zge_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_zge_norm (f16uac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }

  /* nag_zge_load (f16thc).
   * Set the remaining elements of the solutions to zero (to give
   * the minimum-norm solutions), Y2 = 0.
   */
  nag_zge_load(order, n - k, nrhs, zero, zero, &B(k + 1, 1), pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zge_load (f16thc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
```

```
  /* nag_zurmrz (f08bxc).
   * Form W = (Z^H)*Y.
   */
  nag_zunmrz(order, Nag_LeftSide, Nag_ConjTrans, n, nrhs, k, n - k, a, pda,
             tau, b, pdb, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_zurmrz (f08bxc).\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].re = B(i, j).re;
      work[jpvt[i - 1] - 1].im = B(i, j).im;
    }
    /* nag_zge_copy (f16tfc).
     * Copy matrix.
     */
    nag_zge_copy(order, Nag_NoTrans, n, 1, work, pdw, &B(1, j), pdb, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_zge_copy (f16tfc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }

  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print least squares solutions.
   */
  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;
  }

  /* 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 % 7 == 6 ? "\n" : " ");

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

  return exit_status;
}

#undef A
#undef B
```

10.2 Program Data

```
nag_ztzrzf (f08bvc) Example Program Data

   5             4             2                         :Values of m, n and nrhs

 ( 0.47,-0.34) (-0.40, 0.54) ( 0.60, 0.01) ( 0.80,-1.02)
 (-0.32,-0.23) (-0.05, 0.20) (-0.26,-0.44) (-0.43, 0.17)
 ( 0.35,-0.60) (-0.52,-0.34) ( 0.87,-0.11) (-0.34,-0.09)
 ( 0.89, 0.71) (-0.45,-0.45) (-0.02,-0.57) ( 1.14,-0.78)
 (-0.19, 0.06) ( 0.11,-0.85) ( 1.44, 0.80) ( 0.07, 1.14) :End of matrix A

 (-1.08,-2.59) ( 2.22, 2.35)
 (-2.61,-1.49) ( 1.62,-1.48)
 ( 3.13,-3.61) ( 1.65, 3.43)
 ( 7.33,-8.01) (-0.98, 3.08)
 ( 9.12, 7.63) (-2.84, 2.78)                              :End of matrix B
```

10.3 Program Results

```
nag_ztzrzf (f08bvc) Example Program Results

Tolerance used to estimate the rank of A
   1.00e-02
Estimated rank of A
     3

 Least squares solution(s)
                      1                  2
 1  ( 1.1669,-3.3224)  (-0.5023, 1.8323)
 2  ( 1.3486, 5.5027)  (-1.4418,-1.6465)
 3  ( 4.1764, 2.3435)  ( 0.2908, 1.4900)
 4  ( 0.6467, 0.0107)  (-0.2453, 0.3951)

Square root(s) of the residual sum(s) of squares
   2.51e-01    8.10e-02
```