# NAG C Library Function Document

# nag_ztr_copy (f16tec)

## 1 Purpose

nag_ztr_copy (f16tec) copies a complex triangular matrix.

## 2 Specification

```
#include <nag.h>
#include <nagf16.h>
void nag_ztr_copy (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
     Nag_DiagType diag, Integer n, const Complex a[], Integer pda, Complex b[],
     Integer pdb, NagError *fail)
```

## 3 Description

nag_ztr_copy (f16tec) performs the triangular matrix copy operations

$$B \leftarrow A, \quad B \leftarrow A^{\mathrm{T}} \quad \text{or} \quad B \leftarrow A^{\mathrm{H}}.$$

where $A$ and $B$ are $n$ by $n$ complex triangular matrices.

## 4 References

The BLAS Technical Forum Standard (2001) *www.netlib.org/blas/blast-forum*

## 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this argument.

*Constraint*: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **uplo** – Nag_UploType *Input*

*On entry*: specifies whether the upper or lower triangular part of $A$ is stored.

**uplo** = **Nag_Upper**

The upper triangular part of $A$ is stored.

**uplo** = **Nag_Lower**

The lower triangular part of $A$ is stored.

*Constraint*: **uplo** = **Nag_Upper** or **Nag_Lower**.

3: **trans** – Nag_TransType *Input*

*On entry*: specifies the operation to be performed.

**trans** = **Nag_NoTrans**

$B \leftarrow A$.

**trans** = **Nag_Trans**

$B \leftarrow A^{\mathrm{T}}$.

**trans** = **Nag_ConjTrans**

$B \leftarrow A^{\mathrm{H}}$.

*Constraint*: **trans** = **Nag_NoTrans**, **Nag_Trans** or **Nag_ConjTrans**.

4: **diag** – Nag_DiagType *Input*

*On entry*: specifies whether $A$ has non-unit or unit diagonal elements.

**diag** = **Nag_NonUnitDiag**

The diagonal elements are stored explicitly.

**diag** = **Nag_UnitDiag**

The diagonal elements are assumed to be 1 and are not referenced.

*Constraint*: **diag** = **Nag_NonUnitDiag** or **Nag_UnitDiag**.

5: **n** – Integer *Input*

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

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

6: **a**[$dim$] – const Complex *Input*

**Note**: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.

If **order** = **Nag_ColMajor**, the $(i,j)$th element of the matrix $A$ is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = **Nag_RowMajor**, the $(i,j)$th element of the matrix $A$ is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

*On entry*: the $n$ by $n$ triangular matrix $A$.

If **uplo** = **Nag_Upper**, $A$ is upper triangular and the elements of the array below the diagonal are not referenced.

If **uplo** = **Nag_Lower**, $A$ is lower triangular and the elements of the array above the diagonal are not referenced.

7: **pda** – Integer *Input*

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

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

8: **b**[$dim$] – Complex *Output*

**Note**: the dimension, *dim*, of the array **b** must be at least $\max(1, \mathbf{pdb} \times \mathbf{n})$.

If **order** = **Nag_ColMajor**, the $(i,j)$th element of the matrix $B$ is stored in $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = **Nag_RowMajor**, the $(i,j)$th element of the matrix $B$ is stored in $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$.

*On exit*: the $n$ by $n$ triangular matrix $B$.

If **uplo** = **Nag_Upper**, $B$ is upper triangular and the elements of the array below the diagonal are not set.

If **uplo** = **Nag_Lower**, $B$ is lower triangular and the elements of the array above the diagonal are not set.

9: **pdb** – Integer *Input*

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

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

10: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.6 of the Essential Introduction).

## 6 Error Indicators and Warnings

**NE_BAD_PARAM**

On entry, argument ⟨*value*⟩ had an illegal value.

**NE_INT**

On entry, **n** = ⟨*value*⟩.
Constraint: **n** $\geq 0$.

**NE_INT_2**

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

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

## 7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of The BLAS Technical Forum Standard (2001)).

## 8 Further Comments

None.

## 9 Example

Initializes a 4 by 4 lower triangular matrix $A$ and copies its conjugate transpose to the upper triangular part of $B$.

### 9.1 Program Text

```
/* nag_ztr_copy (f16tec) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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

int main(void)
{

  /* Scalars */
  Complex alpha, diag;
```

```
  Integer exit_status, n, pda, pdb;

  /* Arrays */
  Complex *a=0, *b=0;
  char nag_enum_arg[40];

  /* Nag Types */
  NagError fail;
  Nag_OrderType order;
  Nag_UploType uplo;
  Nag_MatrixType matrix;

#ifdef NAG_COLUMN_MAJOR
  order = Nag_ColMajor;
#else
  order = Nag_RowMajor;
#endif

  exit_status = 0;
  INIT_FAIL(fail);

  Vprintf( "nag_ztr_copy (f16tec) Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  /* Read the problem dimension */
  Vscanf("%ld%*[^\n] ", &n);

  /* Read the uplo parameter */
  Vscanf("%s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value(x04nac).
   * Converts NAG enum member name to value
   */
  uplo = nag_enum_name_to_value(nag_enum_arg);

  /* Read scalar parameters */
  Vscanf(" ( %lf , %lf ) ( %lf , %lf )%*[^\n] ",
         &alpha.re, &alpha.im, &diag.re, &diag.im);

  pda = n;
  pdb = n;

  if (n > 0)
    {

      /* Allocate memory */
      if ( !(a = NAG_ALLOC(n*n, Complex)) ||
    !(b = NAG_ALLOC(n*n, Complex)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      Vprintf("Invalid n\n");
      exit_status = 1;
      return exit_status;
    }

  /* nag_ztr_load(f16tgc).
   * Initialize complex triangular matrix.
   *
   */
  nag_ztr_load(order, uplo, n, alpha, diag, a, pda, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_ztr_laod.\n%s\n", fail.message);
      exit_status = 1;
```

```
      goto END;
    }

  /* nag_ztr_copy(f16tec).
   * Copies a complex triangular matrix.
   *
   */
  nag_ztr_copy(order, uplo, Nag_ConjTrans, Nag_NonUnitDiag, n, a, pda, b, pdb,
               &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_ztr_copy.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  if (uplo == Nag_Upper)
    {
      matrix = Nag_LowerMatrix;
    }
  else
    {
      matrix = Nag_UpperMatrix;
    }

  /* Print generated matrix A */
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  nag_gen_complx_mat_print_comp(order, matrix, Nag_NonUnitDiag, n, n, b, pdb,
                                Nag_BracketForm, "%5.2f", "Copied Matrix B",
                                Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
                                0, 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s"
              "\n",fail.message);
      exit_status = 1;
      goto END;
    }

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

  return exit_status;
}
```

## 9.2 Program Data

```
nag_ztr_copy (f16tec) Example Program Data
  4                        : n the dimension of matrix A
  Nag_Lower                : uplo
  ( 0.5,-0.3) ( 9.0, 0.0)  : alpha, diag
```

## 9.3 Program Results

```
nag_ztr_copy (f16tec) Example Program Results

 Copied Matrix B

                1              2              3              4
 1  ( 9.00,-0.00)  ( 0.50, 0.30)  ( 0.50, 0.30)  ( 0.50, 0.30)
 2                 ( 9.00,-0.00)  ( 0.50, 0.30)  ( 0.50, 0.30)
 3                                ( 9.00,-0.00)  ( 0.50, 0.30)
 4                                               ( 9.00,-0.00)
```