

# NAG Library Function Document

## nag\_ztrsm (f16zjc)

### 1 Purpose

nag\_ztrsm (f16zjc) solves a system of equations given as a complex triangular matrix with multiple right-hand sides.

### 2 Specification

```
#include <nag.h>
#include <nagf16.h>

void nag_ztrsm (Nag_OrderType order, Nag_SideType side, Nag_UploType uplo,
               Nag_TransType trans, Nag_DiagType diag, Integer m, Integer n,
               Complex alpha, const Complex a[], Integer pda, Complex b[], Integer pdb,
               NagError *fail)
```

### 3 Description

nag\_ztrsm (f16zjc) performs one of the matrix-matrix operations

$$\begin{array}{l} B \leftarrow \alpha A^{-1}B, \quad B \leftarrow \alpha A^{-T}B, \quad B \leftarrow \alpha A^{-H}B, \\ B \leftarrow \alpha BA^{-1}, \quad B \leftarrow \alpha BA^{-T} \quad \text{or} \quad B \leftarrow \alpha BA^{-H}, \end{array}$$

where  $A$  is a complex triangular matrix,  $B$  is an  $m$  by  $n$  complex matrix, and  $\alpha$  is a complex scalar.  $A^{-T}$  denotes  $A^{-T}$  or equivalently  $A^{-T}$ ;  $A^{-H}$  denotes  $(A^H)^{-1}$  or equivalently  $(A^{-1})^H$ .

### 4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee <http://www.netlib.org/blas/blast-forum/blas-report.pdf>

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

*Constraint:* **order** = Nag\_RowMajor or Nag\_ColMajor.

2: **side** – Nag\_SideType *Input*

*On entry:* specifies whether  $B$  is operated on from the left or the right.

**side** = Nag\_LeftSide

$B$  is pre-multiplied from the left.

**side** = Nag\_RightSide

$B$  is post-multiplied from the right.

*Constraint:* **side** = Nag\_LeftSide or Nag\_RightSide.

- 3: **uplo** – Nag\_UploType *Input*  
*On entry:* specifies whether  $A$  is upper or lower triangular.  
**uplo** = Nag\_Upper  
 $A$  is upper triangular.  
**uplo** = Nag\_Lower  
 $A$  is lower triangular.  
*Constraint:* **uplo** = Nag\_Upper or Nag\_Lower.
- 4: **trans** – Nag\_TransType *Input*  
*On entry:* specifies the operation to be performed.  
**trans** = Nag\_Trans and **side** = Nag\_LeftSide  
 $B \leftarrow \alpha A^{-T} B$ .  
**trans** = Nag\_NoTrans and **side** = Nag\_LeftSide  
 $B \leftarrow \alpha A^{-1} B$ .  
**trans** = Nag\_ConjTrans and **side** = Nag\_LeftSide  
 $B \leftarrow \alpha A^{-H} B$ .  
**trans** = Nag\_Trans and **side** = Nag\_RightSide  
 $B \leftarrow \alpha B A^{-T}$ .  
**trans** = Nag\_NoTrans and **side** = Nag\_RightSide  
 $B \leftarrow \alpha B A^{-1}$ .  
**trans** = Nag\_ConjTrans and **side** = Nag\_RightSide  
 $B \leftarrow \alpha B A^{-H}$ .  
*Constraints:*  
**side** = Nag\_LeftSide or Nag\_RightSide;  
**trans** = Nag\_NoTrans or Nag\_Trans.
- 5: **diag** – Nag\_DiagType *Input*  
*On entry:* specifies whether  $A$  has nonunit 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.
- 6: **m** – Integer *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $B$ ; the order of  $A$  if **side** = Nag\_LeftSide.  
*Constraint:*  $m \geq 0$ .
- 7: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $B$ ; the order of  $A$  if **side** = Nag\_RightSide.  
*Constraint:*  $n \geq 0$ .
- 8: **alpha** – Complex *Input*  
*On entry:* the scalar  $\alpha$ .

- 9: **a**[*dim*] – const Complex *Input*
- Note:** the dimension, *dim*, of the array **a** must be at least
- $\max(1, \mathbf{pda} \times \mathbf{m})$  when **side** = Nag\_LeftSide;  
 $\max(1, \mathbf{pda} \times \mathbf{n})$  when **side** = Nag\_RightSide.
- On entry:* the triangular matrix *A*; *A* is *m* by *m* if **side** = Nag\_LeftSide, or *n* by *n* if **side** = Nag\_RightSide.
- If **order** = 'Nag-ColMajor',  $A_{ij}$  is stored in **a**[(*j* – 1) × **pda** + *i* – 1].
- If **order** = 'Nag-RowMajor',  $A_{ij}$  is stored in **a**[(*i* – 1) × **pda** + *j* – 1].
- If **uplo** = 'Nag-Upper', *A* is upper triangular and the elements of the array corresponding to the lower triangular part of *A* are not referenced.
- If **uplo** = 'Nag-Lower', *A* is lower triangular and the elements of the array corresponding to the upper triangular part of *A* are not referenced.
- If **diag** = 'Nag-UnitDiag', the diagonal elements of *A* are assumed to be 1, and are not referenced.
- 10: **pda** – Integer *Input*
- On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix *A* in the array **a**.
- Constraints:*
- if **side** = Nag\_LeftSide, **pda** ≥ max(1, **m**);  
 if **side** = Nag\_RightSide, **pda** ≥ max(1, **n**).
- 11: **b**[*dim*] – Complex *Input/Output*
- Note:** the dimension, *dim*, of the array **b** must be at least
- $\max(1, \mathbf{pdb} \times \mathbf{n})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{m} \times \mathbf{pdb})$  when **order** = Nag\_RowMajor.
- If **order** = 'Nag-ColMajor',  $B_{ij}$  is stored in **b**[(*j* – 1) × **pdb** + *i* – 1].
- If **order** = 'Nag-RowMajor',  $B_{ij}$  is stored in **b**[(*i* – 1) × **pdb** + *j* – 1].
- On entry:* the *m* by *n* matrix *B*.
- If **alpha** = 0, **b** need not be set.
- On exit:* the updated matrix *B*.
- 12: **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, **m**);  
 if **order** = Nag\_RowMajor, **pdb** ≥ max(1, **n**).
- 13: **fail** – NagError \* *Input/Output*
- The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

**NE\_BAD\_PARAM**

On entry, argument  $\langle value \rangle$  had an illegal value.

**NE\_ENUM\_INT\_2**

On entry, **side** =  $\langle value \rangle$ , **pda** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$ .  
 Constraint: if **side** = Nag\_LeftSide, **pda**  $\geq$  max(1, **m**).

On entry, **side** =  $\langle value \rangle$ , **pda** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .  
 Constraint: if **side** = Nag\_RightSide, **pda**  $\geq$  max(1, **n**).

**NE\_INT**

On entry, **m** =  $\langle value \rangle$ .  
 Constraint: **m**  $\geq$  0.

On entry, **n** =  $\langle value \rangle$ .  
 Constraint: **n**  $\geq$  0.

**NE\_INT\_2**

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

On entry, **pdb** =  $\langle value \rangle$ ; **m** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $\geq$  max(1, **m**).

On entry, **pdb** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $\geq$  max(1, **m**).

On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
 Constraint: **pdb**  $\geq$  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.

**7 Accuracy**

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

**8 Parallelism and Performance**

nag\_ztrsm (f16zjc) is not threaded by NAG in any implementation.

nag\_ztrsm (f16zjc) 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 Users' Note for your implementation for any additional implementation-specific information.

**9 Further Comments**

No test for singularity or near-singularity of  $A$  is included in nag\_ztrsm (f16zjc). Such tests must be performed before calling this function.

## 10 Example

Premultiply complex 4 by 2 matrix  $B$  by inverse of lower triangular 4 by 4 matrix  $A$ ,  $B \leftarrow A^{-1}B$  (or solve  $AX = B$  and return result in  $B$ ), where

$$A = \begin{pmatrix} 4.78 + 4.56i & & & \\ 2.00 - 0.30i & -4.11 + 1.25i & & \\ 2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & \\ -1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i \end{pmatrix}$$

and

$$B = \begin{pmatrix} -14.78 - 32.36i & -18.02 + 28.46i \\ 2.98 - 2.14i & 14.22 + 15.42i \\ -20.96 + 17.06i & 5.62 + 35.89i \\ 9.54 + 9.91i & -16.46 - 1.73i \end{pmatrix}.$$

### 10.1 Program Text

```

/* nag_ztrsm (f16zjc) 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;
    Integer      exit_status, i, j, m, n, pda, pdb;

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

    /* Nag Types */
    NagError     fail;
    Nag_SideType side;
    Nag_DiagType diag;
    Nag_OrderType order;
    Nag_TransType trans;
    Nag_UploType uplo;

#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

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_ztrsm (f16zjc) Example Program Results\n\n");

    /* Skip heading in data file */
    scanf("%*[^\\n] ");
    /* Read the problem dimensions */

```

```

scanf("%ld%ld%*[\n] ", &m, &n);

#ifdef NAG_COLUMN_MAJOR
    pdb = m;
#else
    pdb = n;
#endif

/* Read side */
scanf("%39s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
side = (Nag_SideType) nag_enum_name_to_value(nag_enum_arg);
/* Read uplo */
scanf("%39s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac), see above. */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
/* Read trans */
scanf("%39s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac), see above. */
trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
/* Read diag */
scanf("%39s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value (x04nac), see above. */
diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);
/* Read scalar parameters */
scanf(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);

if (side == Nag_LeftSide)
{
    pda = m;
}
else
{
    pda = n;
}

if (n > 0)
{
    /* Allocate memory */
    if (!(a = NAG_ALLOC(pda*pda, Complex)) ||
        !(b = NAG_ALLOC(n*m, Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}

/* Read A from data file */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= pda; ++i)
    {
        for (j = i; j <= pda; ++j)
            scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    }
    scanf("%*[\n] ");
}
else
{
    for (i = 1; i <= pda; ++i)
    {
        for (j = 1; j <= i; ++j)

```

```

        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
    }
    scanf("%*[\n] ");
}

/* Input matrix B */
for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= n; ++j)
            scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
    }

/* nag_ztrsm (f16zjc).
 * Multiply matrix by inverse of Triangular complex matrix.
 *
 */
nag_ztrsm(order, side, uplo, trans, diag, m, n, alpha, 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;
    }

/* Print the updated matrix B */
/* nag_gen_complx_mat_print_comp (x04dbc).
 * Print complex general matrix (comprehensive)
 */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                              m, n, b, pdb, Nag_BracketForm, "%5.1f",
                              "Updated Matrix B", 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;
    }
END:
NAG_FREE(a);
NAG_FREE(b);

return exit_status;
}

```

## 10.2 Program Data

```

nag_ztrsm (f16zjc) Example Program Data
 4 2 :Values of m and n
Nag_LeftSide :Value of side
Nag_Lower :Value of uplo
Nag_NoTrans :Value of trans
Nag_NonUnitDiag :Value of diag
( 1.00, 0.00) :Value of alpha
( 4.78, 4.56)
( 2.00,-0.30) (-4.11, 1.25)
( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)
(-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26) :End of matrix A
(-14.78,-32.36) (-18.02, 28.46)
( 2.98, -2.14) ( 14.22, 15.42)
(-20.96, 17.06) ( 5.62, 35.89)
( 9.54, 9.91) (-16.46, -1.73) :End of matrix B

```

### 10.3 Program Results

nag\_ztrsm (f16zjc) Example Program Results

Updated Matrix B

	1	2
1	( -5.0, -2.0)	( 1.0, 5.0)
2	( -3.0, -1.0)	( -2.0, -2.0)
3	( 2.0, 1.0)	( 3.0, 4.0)
4	( 4.0, 3.0)	( 4.0, -3.0)

---