

NAG C Library Function Document

nag_ztrmm (f16zfc)

1 Purpose

nag_ztrmm (f16zfc) performs matrix-matrix multiplication for a complex triangular matrix.

2 Specification

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

```
void nag_ztrmm (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_ztrmm (f16zfc) performs one of the matrix-matrix operations

$$\begin{aligned} B &\leftarrow \alpha AB, & B &\leftarrow \alpha A^T B, & B &\leftarrow \alpha A^H B, \\ B &\leftarrow \alpha BA, & B &\leftarrow \alpha BA^T & \text{or} & B &\leftarrow \alpha BA^H, \end{aligned}$$

where B is an m by n complex matrix, A is a complex triangular matrix, and α is a complex scalar.

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: **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 whether the operation involves *A*, A^T or A^H .
trans = Nag_NoTrans
 It involves *A*.
trans = Nag_Trans
 It involves A^T .
trans = Nag_ConjTrans
 It involves A^H .
Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.
- 5: **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.
- 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** ≥ 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** ≥ 0.
- 8: **alpha** – Complex *Input*
On entry: the scalar α .
- 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.
 If **order** = Nag_ColMajor, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*j* – 1) × **pda** + *i* – 1].
 If **order** = Nag_RowMajor, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*i* – 1) × **pda** + *j* – 1].
On entry: the triangular matrix *A*; *A* is *m* by *m* if **side** = Nag_LeftSide, or *n* by *n* if **side** = Nag_RightSide.
 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.

If **diag** = **Nag_UnitDiag**, the diagonal elements of A are not referenced, but are assumed to be 1.

10: **pda** – Integer *Input*

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

Constraints:

if **side** = **Nag_LeftSide**, **pda** \geq $\max(1, \mathbf{m})$;
if **side** = **Nag_RightSide**, **pda** \geq $\max(1, \mathbf{n})$.

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**;
 $\max(1, \mathbf{pdb} \times \mathbf{m})$ when **order** = **Nag_RowMajor**.

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

If **order** = **Nag_RowMajor**, the (i, j)th element of the matrix B is stored in **b**[($i - 1$) \times **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 matrix row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

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

13: **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 $\langle value \rangle$ had an illegal value.

NE_ENUM_INT_2

On entry, **side** = $\langle value \rangle$, **m** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

Constraint: if **side** = **Nag_LeftSide**, **pda** \geq $\max(1, \mathbf{m})$.

On entry, **side** = $\langle value \rangle$, **n** = $\langle value \rangle$, **pda** = $\langle value \rangle$.

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

NE_INT

On entry, **m** = $\langle value \rangle$.

Constraint: **m** \geq 0.

On entry, **n** = $\langle value \rangle$.

Constraint: **n** \geq 0.


```

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);
    Vprintf( "nag_ztrmm (f16zfc) Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n] ");
    /* Read the problem dimensions */
    Vscanf("%ld%ld%*[\n] ", &m, &n);

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

    /* Read side */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac).
     * Converts NAG enum member name to value
     */
    side = nag_enum_name_to_value(nag_enum_arg);
    /* Read uplo */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac), see above. */
    uplo = nag_enum_name_to_value(nag_enum_arg);
    /* Read trans */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac), see above. */
    trans = nag_enum_name_to_value(nag_enum_arg);
    /* Read diag */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac), see above. */
    diag = nag_enum_name_to_value(nag_enum_arg);
    /* Read scalar parameters */
    Vscanf(" ( %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)) )
        {
            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else

```

```

    {
        Vprintf("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)
                    Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            }
        Vscanf("%*[\n] ");
    }
else
    {
        for (i = 1; i <= pda; ++i)
            {
                for (j = 1; j <= i; ++j)
                    Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
            }
        Vscanf("%*[\n] ");
    }

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

/* nag_ztrmm(f16zfc).
 * Triangular complex matrix-matrix multiply.
 *
 */
nag_ztrmm(order, side, uplo, trans, diag, m, n, alpha, a, pda,
          b, pdb, &fail);
if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from ztrmm.\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)
 */
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                              m, n, b, pdb, Nag_BracketForm, "%7.4f",
                              "Updated 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_ztrmm (f16zfc) 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
  (-5.00,-2.00) ( 1.00, 5.00)
  (-3.00,-1.00) (-2.00,-2.00)
  ( 2.00, 1.00) ( 3.00, 4.00)
  ( 4.00, 3.00) ( 4.00,-3.00)                               :End of matrix B
```

9.3 Program Results

nag_ztrmm (f16zfc) Example Program Results

Updated Matrix B

	1	2
1	(-14.7800,-32.3600)	(-18.0200,28.4600)
2	(2.9800,-2.1400)	(14.2200,15.4200)
3	(-20.9600,17.0600)	(5.6200,35.8900)
4	(9.5400, 9.9100)	(-16.4600,-1.7300)
