NAG 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

$$\begin{array}{lll} B \leftarrow \alpha AB, & B \leftarrow \alpha A^{\mathrm{T}}B, & B \leftarrow \alpha A^{\mathrm{H}}B, \\ B \leftarrow \alpha BA, & B \leftarrow \alpha BA^{\mathrm{T}} \quad \text{or} & B \leftarrow \alpha BA^{\mathrm{H}}, \end{array}$$

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

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 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 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: $\mathbf{m} \geq 0$.

7: **n** – Integer *Input*

On entry: n, the number of columns of the matrix B; the order of A if **side** = Nag_RightSide.

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

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.

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 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = 'Nag_RowMajor', A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

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

If **order** = 'Nag_ColMajor', B_{ij} is stored in **b**$[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = 'Nag_RowMajor', B_{ij} is stored in **b**$[(i-1) \times \mathbf{pdb} + j - 1]$.

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

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

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

NE_ENUM_INT_2

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

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

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

NE_INT_2

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

On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

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.

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

Not applicable.

9 Further Comments

None.

10 Example

Premultiply complex 4 by 2 matrix B by lower triangular 4 by 4 matrix A, $B \leftarrow AB$, 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

$$B = \begin{pmatrix} -5.0 - 2.0i & 1.0 + 5.0i \\ -3.0 - 1.0i & -2.0 - 2.0i \\ 2.0 + 1.0i & 3.0 + 4.0i \\ 4.0 + 3.0i & 4.0 - 3.0i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_ztrmm (f16zfc) 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_ztrmm (f16zfc) 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_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)
    {
      printf("Error from nag_ztrmm (f16zfc).\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, "%7.4f",
                                "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_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
```

10.3 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)
```