NAG Library Function Document

nag_ztrmm (f16zfc)

1 Purpose

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

2 Specification

```c
#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

\[
B \leftarrow \alpha AB, \quad B \leftarrow \alpha A^TB, \quad B \leftarrow \alpha A^H B,
\]

\[
B \leftarrow \alpha BA, \quad B \leftarrow \alpha BA^T \quad \text{or} \quad B \leftarrow \alpha BA^H,
\]

where \( B \) is an \( m \) by \( n \) complex matrix, \( A \) is a complex triangular matrix, and \( \alpha \) is a complex scalar.

4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) Basic Linear Algebra

5 Arguments

1: \( \text{order} \) – Nag_OrderType \hspace{1cm} \text{Input}

\( \text{On entry}: \) the \( \text{order} \) 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
\( \text{order} = \text{Nag_RowMajor}. \) See Section 3.2.1.3 in the Essential Introduction for a more detailed
explanation of the use of this argument.

\( \text{Constraint: } \text{order} = \text{Nag_RowMajor or Nag_ColMajor}. \)

2: \( \text{side} \) – Nag_SideType \hspace{1cm} \text{Input}

\( \text{On entry}: \) specifies whether \( B \) is operated on from the left or the right.

\( \text{side} = \text{Nag_LeftSide} \)
\( B \) is pre-multiplied from the left.

\( \text{side} = \text{Nag_RightSide} \)
\( B \) is post-multiplied from the right.

\( \text{Constraint: } \text{side} = \text{Nag_LeftSide or Nag_RightSide}. \)
3: \texttt{uplo} – Nag_UploType

\textit{Input}

\textit{On entry:} specifies whether \(A\) is upper or lower triangular.

\texttt{uplo} = Nag_Upper
\newline \hspace{1em} \(A\) is upper triangular.

\texttt{uplo} = Nag_Lower
\newline \hspace{1em} \(A\) is lower triangular.

\textit{Constraint:} \texttt{uplo} = Nag_Upper or Nag_Lower.

4: \texttt{trans} – Nag_TransType

\textit{Input}

\textit{On entry:} specifies whether the operation involves \(A\), \(A^T\) or \(A^H\).

\texttt{trans} = Nag_NoTrans
\newline \hspace{1em} It involves \(A\).

\texttt{trans} = Nag_Trans
\newline \hspace{1em} It involves \(A^T\).

\texttt{trans} = Nag_ConjTrans
\newline \hspace{1em} It involves \(A^H\).

\textit{Constraint:} \texttt{trans} = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

5: \texttt{diag} – Nag_DiagType

\textit{Input}

\textit{On entry:} specifies whether \(A\) has nonunit or unit diagonal elements.

\texttt{diag} = Nag_NonUnitDiag
\newline \hspace{1em} The diagonal elements are stored explicitly.

\texttt{diag} = Nag_UnitDiag
\newline \hspace{1em} The diagonal elements are assumed to be 1 and are not referenced.

\textit{Constraint:} \texttt{diag} = Nag_NonUnitDiag or Nag_UnitDiag.

6: \texttt{m} – Integer

\textit{Input}

\textit{On entry:} \(m\), the number of rows of the matrix \(B\); the order of \(A\) if \texttt{side} = Nag_LeftSide.

\textit{Constraint:} \(m \geq 0\).

7: \texttt{n} – Integer

\textit{Input}

\textit{On entry:} \(n\), the number of columns of the matrix \(B\); the order of \(A\) if \texttt{side} = Nag_RightSide.

\textit{Constraint:} \(n \geq 0\).

8: \texttt{alpha} – Complex

\textit{Input}

\textit{On entry:} the scalar \(\alpha\).

9: \texttt{a[dim]} – const Complex

\textit{Input}

\textit{Note:} the dimension, \texttt{dim}, of the array \(a\) must be at least

\[
\max(1, \texttt{pda} \times \texttt{m}) \text{ when } \texttt{side} = \text{Nag_LeftSide}; \\
\max(1, \texttt{pda} \times \texttt{n}) \text{ when } \texttt{side} = \text{Nag_RightSide}.
\]

\textit{On entry:} the triangular matrix \(A\); \(A\) is \(m\) by \(m\) if \texttt{side} = Nag_LeftSide, or \(n\) by \(n\) if \texttt{side} = Nag_RightSide.

If \texttt{order} = Nag_ColMajor, \(A_{ij}\) is stored in \(a[(j - 1) \times \texttt{pda} + i - 1]\).

If \texttt{order} = Nag_RowMajor, \(A_{ij}\) is stored in \(a[(i - 1) \times \texttt{pda} + j - 1]\).
If $\text{uplo} = \text{Nag\_Upper}$, $A$ is upper triangular and the elements of the array corresponding to the lower triangular part of $A$ are not referenced.

If $\text{uplo} = \text{Nag\_Lower}$, $A$ is lower triangular and the elements of the array corresponding to the upper triangular part of $A$ are not referenced.

If $\text{diag} = \text{Nag\_UnitDiag}$, the diagonal elements of $A$ are assumed to be 1, and are not referenced.

10: \textbf{pda} – Integer \hspace{2cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) of the matrix $A$ in the array \texttt{a}.

\textit{Constraints:}
\begin{itemize}
  \item if $\text{side} = \text{Nag\_LeftSide}$, \texttt{pda} $\geq \max(1, \text{m})$;
  \item if $\text{side} = \text{Nag\_RightSide}$, \texttt{pda} $\geq \max(1, \text{n})$.
\end{itemize}

11: \textbf{b[\textit{dim}]} – Complex \hspace{2cm} \textit{Input/Output}

\textit{Note:} the dimension, \textit{dim}, of the array \texttt{b} must be at least \begin{itemize}
  \item $\max(1, \text{pdb} \times \text{n})$ when \texttt{order} = \texttt{Nag\_ColMajor};
  \item $\max(1, \text{m} \times \text{pdb})$ when \texttt{order} = \texttt{Nag\_RowMajor}.
\end{itemize}

If \texttt{order} = \texttt{Nag\_ColMajor}, $B_{ij}$ is stored in $\texttt{b}[(j - 1) \times \text{pdb} + i - 1]$.

If \texttt{order} = \texttt{Nag\_RowMajor}, $B_{ij}$ is stored in $\texttt{b}[(i - 1) \times \text{pdb} + j - 1]$.

\textit{On entry:} the $m$ by $n$ matrix $B$.

\textit{If \texttt{alpha} = 0, \texttt{b} need not be set.}

\textit{On exit:} the updated matrix $B$.

12: \textbf{pdb} – Integer \hspace{2cm} \textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{b}.

\textit{Constraints:}
\begin{itemize}
  \item if \texttt{order} = \texttt{Nag\_ColMajor}, \texttt{pdb} $\geq \max(1, \text{m})$;
  \item if \texttt{order} = \texttt{Nag\_RowMajor}, \texttt{pdb} $\geq \max(1, \text{n})$.
\end{itemize}

13: \textbf{fail} – Nag\_Error * \hspace{2cm} \textit{Input/Output}

\textit{The NAG error argument (see Section 3.6 in the Essential Introduction).}

6 \quad \textbf{Error Indicators and Warnings}

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

On entry, argument \texttt{\langle value\rangle} had an illegal value.

\textbf{NE\_ENUM\_INT\_2}

On entry, \texttt{side} = \texttt{\langle value\rangle}, \texttt{m} = \texttt{\langle value\rangle}, \texttt{pda} = \texttt{\langle value\rangle}.

Constraint: if \texttt{side} = \texttt{Nag\_LeftSide}, \texttt{pda} $\geq \max(1, \text{m})$.

On entry, \texttt{side} = \texttt{\langle value\rangle}, \texttt{n} = \texttt{\langle value\rangle}, \texttt{pda} = \texttt{\langle value\rangle}.

Constraint: if \texttt{side} = \texttt{Nag\_RightSide}, \texttt{pda} $\geq \max(1, \text{n})$. 
NE_INT
On entry, $m = \langle\text{value}\rangle$.
Constraint: $m \geq 0$.
On entry, $n = \langle\text{value}\rangle$.
Constraint: $n \geq 0$.

NE_INT_2
On entry, $pdb = \langle\text{value}\rangle$, $m = \langle\text{value}\rangle$.
Constraint: $pdb \geq \max(1, m)$.
On entry, $pdb = \langle\text{value}\rangle$ and $n = \langle\text{value}\rangle$.
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.
An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction 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 the Essential Introduction for further information.

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

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

and

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

    if (NAG_COLUMN_MAJOR)
        #define A(I, J) a[(J-1)*pda + I - 1]
        #define B(I, J) b[(J-1)*pdb + I - 1]
    else
        #define A(I, J) a[(I-1)*pda+J-1]
        #define B(I, J) b[(I-1)*pdb + J - 1]

    order = Nag_ColMajor;
    #endif

    exit_status = 0;
    INIT_FAIL(fail);

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

    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*
         ");
    #else
        scanf("%*
         ");
    #endif

    /* Read the problem dimensions */
    #ifdef _WIN32
        scanf("%\n", &m, &n);
    #else
        scanf("%\n", &m, &n);
    #endif

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

    /* Read side */
    #ifdef _WIN32
        scanf("%s\n", nag_enum_arg, __countof(nag_enum_arg));
    #else
        scanf("%s\n", nag_enum_arg, __countof(nag_enum_arg));
    #endif
}
#else
   scanf("%39s\n", nag_enum_arg);
#endif /* nag_enum_name_to_value (x04nac). */
   side = (Nag_SideType) nag_enum_name_to_value(nag_enum_arg);
/* Read uplo */
#ifdef _WIN32
   scanf_s("%39s\n", nag_enum_arg, _countof(nag_enum_arg));
#else
   scanf("%39s\n", nag_enum_arg);
#endif /* nag_enum_name_to_value (x04nac), see above. */
   uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
/* Read trans */
#ifdef _WIN32
   scanf_s("%39s\n", nag_enum_arg, _countof(nag_enum_arg));
#else
   scanf("%39s\n", nag_enum_arg);
#endif /* nag_enum_name_to_value (x04nac), see above. */
   trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
/* Read diag */
#ifdef _WIN32
   scanf_s("%39s\n", nag_enum_arg, _countof(nag_enum_arg));
#else
   scanf("%39s\n", nag_enum_arg);
#endif /* nag_enum_name_to_value (x04nac), see above. */
   diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);
/* Read scalar parameters */
#ifdef _WIN32
   scanf_s(" ( %lf , %lf )\n", &alpha.re, &alpha.im);
#else
   scanf(" ( %lf , %lf )\n", &alpha.re, &alpha.im);
#endif

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)
```c
#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
#endif _WIN32

/* Input matrix B */
for (i = 1; i <= m; ++i)
{
    for (j = 1; j <= n; ++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
    }
}

/* 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", 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", NagIntegerLabels,
    0, NagIntegerLabels, 0, 80, 0, 0,
    &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n", fail.message);
    exit_status = 1;
    goto END;
}
```

Mark 25

f16zfc
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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-14.7800,-32.3600)</td>
<td>(-18.0200,28.4600)</td>
</tr>
<tr>
<td>2</td>
<td>( 2.9800,-2.1400)</td>
<td>(14.2200,15.4200)</td>
</tr>
<tr>
<td>3</td>
<td>(-20.9600,17.0600)</td>
<td>( 5.6200,35.8900)</td>
</tr>
<tr>
<td>4</td>
<td>( 9.5400, 9.9100)</td>
<td>(-16.4600,-1.7300)</td>
</tr>
</tbody>
</table>