

# NAG C Library Function Document

## nag\_ztbmv (f16sgc)

### 1 Purpose

nag\_ztbmv (f16sgc) performs matrix-vector multiplication for a complex triangular band matrix.

### 2 Specification

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

```
void nag_ztbmv (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
               Nag_DiagType diag, Integer n, Integer k, Complex alpha, const Complex ab[],
               Integer pdab, Complex x[], Integer incx, NagError *fail)
```

### 3 Description

nag\_ztbmv (f16sgc) performs one of the matrix-vector operations

$$x \leftarrow \alpha Ax, \quad x \leftarrow \alpha A^T x \quad \text{or} \quad x \leftarrow \alpha A^H x,$$

where  $A$  is an  $n$  by  $n$  complex triangular band matrix with  $k$  subdiagonals or superdiagonals, and  $x$  is an  $n$  element complex vector and  $\alpha$  is a complex scalar.

### 4 References

The BLAS Technical Forum Standard (2001) [www.netlib.org/blas/blast-forum](http://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  $A$  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**.
- 3: **trans** – Nag\_TransType *Input*  
*On entry:* specifies the operation to be performed.  
**trans** = **Nag\_NoTrans**  
 $x \leftarrow \alpha Ax$ .

**trans** = Nag\_Trans

$$x \leftarrow \alpha A^T x.$$

**trans** = Nag\_ConjTrans

$$x \leftarrow \alpha A^H x.$$

*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 matrix  $A$ .

*Constraint:*  $n \geq 0$ .

6: **k** – Integer *Input*

*On entry:*  $k$ , the number of subdiagonals or superdiagonals of the matrix  $A$ .

*Constraint:*  $k \geq 0$ .

7: **alpha** – Complex *Input*

*On entry:* the scalar  $\alpha$ .

8: **ab**[*dim*] – const Complex *Input*

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

*On entry:* the  $n$  by  $n$  triangular band matrix  $A$ . This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements  $a_{ij}$  depends on the **order** and **uplo** arguments as follows:

if **order** = Nag\_ColMajor and **uplo** = Nag\_Upper,  
 $a_{ij}$  is stored in **ab**[ $k + i - j + (j - 1) \times \mathbf{pdab}$ ],  
for  $j = 1, \dots, n$  and  $i = \max(1, j - k), \dots, j$ ;  
if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $a_{ij}$  is stored in **ab**[ $i - j + (j - 1) \times \mathbf{pdab}$ ],  
for  $j = 1, \dots, n$  and  $i = j, \dots, \min(n, j + k)$ ;  
if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $a_{ij}$  is stored in **ab**[ $j - i + (i - 1) \times \mathbf{pdab}$ ],  
for  $i = 1, \dots, n$  and  $j = i, \dots, \min(n, i + k)$ ;  
if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $a_{ij}$  is stored in **ab**[ $k + j - i + (i - 1) \times \mathbf{pdab}$ ],  
for  $i = 1, \dots, n$  and  $j = \max(1, i - k), \dots, i$ .

9: **pdab** – Integer *Input*

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

*Constraint:*  $\mathbf{pdab} \geq \mathbf{k} + 1$ .

- 10: **x**[*dim*] – Complex Input/Output  
**Note:** the dimension, *dim*, of the array **x** must be at least  $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$ .  
*On entry:* the right-hand side vector *b*.  
*On exit:* the solution vector *x*.
- 11: **incx** – Integer Input  
*On entry:* the increment in the subscripts of **x** between successive elements of *x*.  
*Constraint:* **incx**  $\neq 0$ .
- 12: **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\_INT

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

Constraint: **incx**  $\neq 0$ .

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

Constraint: **k**  $\geq 0$ .

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

Constraint: **n**  $\geq 0$ .

### NE\_INT\_2

On entry, **pdab** =  $\langle value \rangle$ , **k** =  $\langle value \rangle$ .

Constraint: **pdab**  $\geq \mathbf{k} + 1$ .

## 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

No test for singularity or near-singularity of *A* is included in nag\_ztbmv (f16sgc). Such tests must be performed before calling this function.

## 9 Example

To compute the matrix-vector product

$$y = \alpha Ax$$

where

$$A = \begin{pmatrix} 1.0 + 1.0i & 0.0 + 0.0i & 0.0 + 0.0i & 0.0 + 0.0i \\ 2.0 + 1.0i & 2.0 + 2.0i & 0.0 + 0.0i & 0.0 + 0.0i \\ 0.0 + 0.0i & 3.0 + 2.0i & 3.0 + 3.0i & 0.0 + 0.0i \\ 0.0 + 0.0i & 0.0 + 0.0i & 4.0 + 3.0i & 4.0 + 4.0i \end{pmatrix},$$

$$x = \begin{pmatrix} 1.0 + 1.0i \\ -2.0 + 2.0i \\ 3.0 - 2.0i \\ -1.0 + 1.0i \end{pmatrix}$$

and

$$\alpha = 1.0 + 0.0i.$$

## 9.1 Program Text

```

/* nag_ztbmv (f16sgc) 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, incx, j, k, kd, n, pdab, xlen;

    /* Arrays */
    Complex *ab=0, *x=0;
    char nag_enum_arg[40];

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

#ifdef NAG_COLUMN_MAJOR
#define AB_UPPER(I,J) ab[(J-1)*pdab + k + I - J - 1]
#define AB_LOWER(I,J) ab[(J-1)*pdab + I - J]
    order = Nag_ColMajor;
#else
#define AB_UPPER(I,J) ab[(I-1)*pdab + J - I]
#define AB_LOWER(I,J) ab[(I-1)*pdab + k + J - I - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);
    Vprintf( "nag_ztbmv (f16sgc) Example Program Results\n\n");

    /* Skip heading in data file */
    Vscanf("%*[\n] ");
    /* Read the problem dimension */
    Vscanf("%ld%ld%*[\n] ", &n, &kd);
    /* Read uplo */
    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 trans */
    Vscanf("%s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value(x04nac).

```

```

    * Converts NAG enum member name to value
    */
trans = nag_enum_name_to_value(nag_enum_arg);
/* Read diag */
Vscanf("%s%*[\n] ", nag_enum_arg);
/* nag_enum_name_to_value(x04nac).
 * Converts NAG enum member name to value
 */
diag = nag_enum_name_to_value(nag_enum_arg);
/* Read scalar parameters */
Vscanf(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
/* Read increment parameters */
Vscanf("%ld%*[\n] ", &incx);

pdab = kd + 1;
xlen = MAX(1, 1 + (n - 1)*ABS(incx));

if (n > 0)
{
    /* Allocate memory */
    if ( !(ab = NAG_ALLOC(pdab*n, Complex)) ||
        !(x = NAG_ALLOC(xlen, 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 */
k = kd + 1;
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= MIN(i+kd,n); ++j)
            Vscanf(" ( %lf , %lf )",
                &AB_UPPER(i,j).re, &AB_UPPER(i,j).im);
    }
    Vscanf("%*[\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = MAX(1,i-kd); j <= i; ++j)
            Vscanf(" ( %lf , %lf )",
                &AB_LOWER(i,j).re, &AB_LOWER(i,j).im);
    }
    Vscanf("%*[\n] ");
}

/* Input vector x */
for (i = 1; i <= xlen; ++i)
    Vscanf(" ( %lf , %lf )%*[\n] ", &x[i - 1].re, &x[i - 1].im);

/* nag_ztbmv(f16sgc).
 * Complex triangular banded matrix-vector multiply.
 */
nag_ztbmv(order, uplo, trans, diag, n, kd, alpha, ab, pdab,
          x, incx, &fail);
if (fail.code != NE_NOERROR)
{

```

```

    Vprintf("Error from ztbmv.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print output vector x */
Vprintf("%s\n", " x");
for (i = 1; i <= xlen; ++i)
{
    Vprintf("(%11f,%11f)\n", x[i-1].re, x[i - 1].im);
}

END:
if (ab) NAG_FREE(ab);
if (x) NAG_FREE(x);

return exit_status;
}

```

## 9.2 Program Data

```

nag_ztbmv (f16sgc) Example Program Data
4 1 :Values of n, kd
Nag_Lower :Value of uplo
Nag_NoTrans :Value of trans
Nag_NonUnitDiag :Value of diag
( 1.0, 0.0) :Value of alpha
1 :Value of incx
( 1.0, 1.0)
( 2.0, 1.0) ( 2.0, 2.0)
( 3.0, 2.0) ( 3.0, 3.0)
( 4.0, 3.0) ( 4.0, 4.0) :End of matrix A

( 1.0, 1.0)
(-2.0, 2.0)
( 3.0,-2.0)
(-1.0, 1.0) :End of vector x

```

## 9.3 Program Results

```

nag_ztbmv (f16sgc) Example Program Results

```

```

x
( 0.000000, 2.000000)
( -7.000000, 3.000000)
( 5.000000, 5.000000)
( 10.000000, 1.000000)

```

---