

NAG 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, x is an n -element complex vector 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 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.
- 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 α .
- 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 of 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 $j = \max(1, i - k), \dots, i$.

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

- 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: **pdab** $\geq k + 1$.
- 10: **x[dim]** – Complex *Input/Output*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (n - 1)|incx|)$.
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 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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** ≥ 0 .

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

NE_INT_2

On entry, **pdab** = $\langle value \rangle$, **k** = $\langle value \rangle$.
Constraint: **pdab** $\geq k + 1$.

NE_INTERNAL_ERROR

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation 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

nag_ztbmv (f16sgc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example computes 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

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

10.1 Program Text

```
/* nag_ztbmv (f16sgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.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);

    printf("nag_ztbmv (f16sgc) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    /* Read the problem dimension */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &n, &kd);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &n, &kd);
#endif
    /* Read uplo */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    /* Read trans */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
    /* Read diag */
#ifdef _WIN32
    scanf_s("%39s%*[\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n] ", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    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
    /* Read increment parameters */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &incx);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &incx);

```

```

#endif

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)))
    {
        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 */
k = kd + 1;
if (uplo == Nag_Upper) {
    for (i = 1; i <= n; ++i) {
        for (j = i; j <= MIN(i + kd, n); ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#else
            scanf(" ( %lf , %lf )", &AB_UPPER(i, j).re, &AB_UPPER(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else {
    for (i = 1; i <= n; ++i) {
        for (j = MAX(1, i - kd); j <= i; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_LOWER(i, j).im);
#else
            scanf(" ( %lf , %lf )", &AB_LOWER(i, j).re, &AB_LOWER(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}

/* Input vector x */
for (i = 1; i <= xlen; ++i)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )%*[\n] ", &x[i - 1].re, &x[i - 1].im);
#else
    scanf(" ( %lf , %lf )%*[\n] ", &x[i - 1].re, &x[i - 1].im);
#endif

/* 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) {
    printf("Error from nag_ztbmv (f16sgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

```

```

}

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

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

    return exit_status;
}

```

10.2 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

```

10.3 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)

```
