

# NAG Library Function Document

## nag\_zgbmv (f16sbc)

### 1 Purpose

nag\_zgbmv (f16sbc) performs matrix-vector multiplication for a complex band matrix.

### 2 Specification

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

void nag_zgbmv (Nag_OrderType order, Nag_TransType trans, Integer m,
               Integer n, Integer kl, Integer ku, Complex alpha, const Complex ab[],
               Integer pdab, const Complex x[], Integer incx, Complex beta,
               Complex y[], Integer incy, NagError *fail)
```

### 3 Description

nag\_zgbmv (f16sbc) performs one of the matrix-vector operations

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

where  $A$  is an  $m$  by  $n$  complex band matrix with  $k_l$  subdiagonals and  $k_u$  superdiagonals,  $x$  and  $y$  are complex vectors, and  $\alpha$  and  $\beta$  are complex scalars.

If  $m = 0$  or  $n = 0$ , no operation is performed.

### 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: **trans** – Nag\_TransType *Input*

*On entry:* specifies the operation to be performed.

**trans** = Nag\_NoTrans

$$y \leftarrow \alpha Ax + \beta y.$$

**trans** = Nag\_Trans

$$y \leftarrow \alpha A^T x + \beta y.$$

**trans** = Nag\_ConjTrans

$$y \leftarrow \alpha A^H x + \beta y.$$

*Constraint:* **trans** = Nag\_NoTrans, Nag\_Trans or Nag\_ConjTrans.

- 3: **m** – Integer *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $A$ .  
*Constraint:*  $\mathbf{m} \geq 0$ .
- 4: **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $A$ .  
*Constraint:*  $\mathbf{n} \geq 0$ .
- 5: **kl** – Integer *Input*  
*On entry:*  $k_l$ , the number of subdiagonals within the band of  $A$ .  
*Constraint:*  $\mathbf{kl} \geq 0$ .
- 6: **ku** – Integer *Input*  
*On entry:*  $k_u$ , the number of superdiagonals within the band of  $A$ .  
*Constraint:*  $\mathbf{ku} \geq 0$ .
- 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  
 $\max(1, \mathbf{pdab} \times \mathbf{n})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{m} \times \mathbf{pdab})$  when **order** = Nag\_RowMajor.  
*On entry:* the  $m$  by  $n$  band matrix  $A$ .  
This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements  $A_{ij}$ , for row  $i = 1, \dots, m$  and column  $j = \max(1, i - k_l), \dots, \min(n, i + k_u)$ , depends on the **order** argument as follows:  
if **order** = Nag\_ColMajor,  $A_{ij}$  is stored as **ab**[( $j - 1$ )  $\times$  **pdab** + **ku** +  $i - j$ ];  
if **order** = Nag\_RowMajor,  $A_{ij}$  is stored as **ab**[( $i - 1$ )  $\times$  **pdab** + **kl** +  $j - i$ ].
- 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:*  $\mathbf{pdab} \geq \mathbf{kl} + \mathbf{ku} + 1$ .
- 10: **x**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **x** must be at least  
 $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$  when **trans** = Nag\_NoTrans;  
 $\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incx}|)$  when **trans** = Nag\_Trans or Nag\_ConjTrans.  
*On entry:* the vector  $x$ .  
If **trans** = Nag\_NoTrans, then  $x$  is an  $n$ -element vector.  
If  $\mathbf{incx} > 0$ ,  $x_i$  must be stored in **x**[( $i - 1$ )  $\times$  **incx**], for  $i = 1, 2, \dots, \mathbf{n}$ .  
If  $\mathbf{incx} < 0$ ,  $x_i$  must be stored in **x**[( $\mathbf{n} - i$ )  $\times$  **incx**], for  $i = 1, 2, \dots, \mathbf{n}$ .  
Intermediate elements of **x** are not referenced. If  $\mathbf{n} = 0$ , **x** is not referenced and may be NULL.

Otherwise,  $x$  is an  $m$ -element vector.

If  $\mathbf{incx} > 0$ ,  $x_i$  must be stored in  $\mathbf{x}[(i-1) \times \mathbf{incx}]$ , for  $i = 1, 2, \dots, \mathbf{m}$ .

If  $\mathbf{incx} < 0$ ,  $x_i$  must be stored in  $\mathbf{x}[(\mathbf{m}-i) \times |\mathbf{incx}|]$ , for  $i = 1, 2, \dots, \mathbf{m}$ .

Intermediate elements of  $\mathbf{x}$  are not referenced. If  $\mathbf{m} = 0$ ,  $\mathbf{x}$  is not referenced and may be **NULL**.

- 11: **incx** – Integer *Input*  
*On entry:* the increment in the subscripts of  $\mathbf{x}$  between successive elements of  $x$ .  
*Constraint:*  $\mathbf{incx} \neq 0$ .
- 12: **beta** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 13: **y[dim]** – Complex *Input/Output*  
**Note:** the dimension,  $\mathit{dim}$ , of the array  $\mathbf{y}$  must be at least  
 $\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incy}|)$  when **trans** = Nag\_NoTrans;  
 $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incy}|)$  when **trans** = Nag\_Trans or Nag\_ConjTrans.  
*On entry:* the vector  $y$ . See  $\mathbf{x}$  for details of storage.  
If **beta** = 0,  $\mathbf{y}$  need not be set.  
*On exit:* the updated vector  $y$ .
- 14: **incy** – Integer *Input*  
*On entry:* the increment in the subscripts of  $\mathbf{y}$  between successive elements of  $y$ .  
*Constraint:*  $\mathbf{incy} \neq 0$ .
- 15: **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  $\langle \mathit{value} \rangle$  had an illegal value.

### NE\_INT

On entry, **incx** =  $\langle \mathit{value} \rangle$ .  
Constraint:  $\mathbf{incx} \neq 0$ .

On entry, **incy** =  $\langle \mathit{value} \rangle$ .  
Constraint:  $\mathbf{incy} \neq 0$ .

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

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

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

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

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

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

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

### NE\_INT\_3

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

Constraint: **pdab**  $\geq$  **kl** + **ku** + 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\_zgbmv (f16sbc) is not threaded in any implementation.

## 9 Further Comments

None.

## 10 Example

This example computes the matrix-vector product

$$y = \alpha Ax + \beta y$$

where

$$A = \begin{pmatrix} 1.0 + 1.0i & 1.0 + 2.0i & 0.0 + 0.0i & 0.0 + 0.0i \\ 2.0 + 1.0i & 2.0 + 2.0i & 2.0 + 3.0i & 0.0 + 0.0i \\ 3.0 + 1.0i & 3.0 + 2.0i & 3.0 + 3.0i & 3.0 + 4.0i \\ 0.0 + 0.0i & 4.0 + 2.0i & 4.0 + 3.0i & 4.0 + 4.0i \\ 0.0 + 0.0i & 0.0 + 0.0i & 5.0 + 3.0i & 5.0 + 4.0i \\ 0.0 + 0.0i & 0.0 + 0.0i & 0.0 + 0.0i & 6.0 + 4.0i \end{pmatrix},$$

$$x = \begin{pmatrix} 1.0 - 1.0i \\ 2.0 - 2.0i \\ 3.0 - 3.0i \\ 4.0 - 4.0i \end{pmatrix},$$

$$y = \begin{pmatrix} -3.5 + 0.0i \\ -11.5 + 1.0i \\ -27.5 + 3.0i \\ -29.0 + 7.5i \\ -25.5 + 10.0i \\ -14.5 + 10.0i \end{pmatrix},$$

$$\alpha = 1.0 + 0.0i \quad \text{and} \quad \beta = 2.0 + 0.0i.$$

## 10.1 Program Text

```

/* nag_zgbmv (f16sbc) 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, beta;
    Integer ab_size, exit_status, i, incx, incy, j, kl, ku;
    Integer m, n, pdab, xlen, ylen;

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

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_TransType trans;

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

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_zgbmv (f16sbc) Example Program Results\n\n");

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

    /* Read the problem dimensions */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n] ",
            &m, &n, &kl, &ku);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[\n] ",

```

```

        &m, &n, &kl, &ku);
#endif
/* Read the transpose parameter */
#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 scalar parameters */
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
            &alpha.re, &alpha.im, &beta.re, &beta.im);
#else
    scanf(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
            &alpha.re, &alpha.im, &beta.re, &beta.im);
#endif
/* Read increment parameters */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[\n] ", &incx, &incy);
#endif

    pdab = kl + ku + 1;
#ifdef NAG_COLUMN_MAJOR
    ab_size = pdab * n;
#else
    ab_size = pdab * m;
#endif

    if (trans == Nag_NoTrans) {
        xlen = MAX(1, 1 + (n - 1) * ABS(incx));
        ylen = MAX(1, 1 + (m - 1) * ABS(incy));
    }
    else {
        xlen = MAX(1, 1 + (m - 1) * ABS(incx));
        ylen = MAX(1, 1 + (n - 1) * ABS(incy));
    }

    if (m > 0 && n > 0) {
        /* Allocate memory */
        if (!(ab = NAG_ALLOC(ab_size, Complex)) ||
            !(x = NAG_ALLOC(xlen, Complex)) || !(y = NAG_ALLOC(ylen, Complex)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else {
        printf("Invalid m or n\n");
        exit_status = 1;
        return exit_status;
    }

    /* Input matrix A and vectors x and y */

    for (i = 1; i <= m; ++i) {
        for (j = MAX(1, i - kl); j <= MIN(n, i + ku); ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#else
            scanf(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else

```

```

        scanf("%*[^\\n] ");
#endif
    }
    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
    for (i = 1; i <= ylen; ++i)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )%*[^\\n] ", &y[i - 1].re, &y[i - 1].im);
    #else
        scanf(" ( %lf , %lf )%*[^\\n] ", &y[i - 1].re, &y[i - 1].im);
    #endif

    /* nag_zgbmv (f16sbc).
     * Complex valued band matrix-vector multiply.
     */
    nag_zgbmv(order, trans, m, n, kl, ku, alpha, ab, pdab, x,
              incx, beta, y, incy, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_zgbmv.\\n%s\\n", fail.message);
        exit_status = 1;
        goto END;
    }

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

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

    return exit_status;
}

```

## 10.2 Program Data

nag\_zgbmv (f16sbc) Example Program Data

```

6 4 2 1                               :Values of m, n, kl, ku
Nag_NoTrans                           : trans
( 1.0, 0.0) ( 2.0, 0.0)               : alpha, beta
1 1                                    : incx, incy
( 1.0, 1.0) ( 1.0, 2.0)
( 2.0, 1.0) ( 2.0, 2.0) ( 2.0, 3.0)
( 3.0, 1.0) ( 3.0, 2.0) ( 3.0, 3.0) ( 3.0, 4.0)
( 4.0, 2.0) ( 4.0, 3.0) ( 4.0, 4.0)
( 5.0, 3.0) ( 5.0, 4.0)
( 6.0, 4.0) : the end of matrix A

( 1.0,-1.0)
( 2.0,-2.0)
( 3.0,-3.0)
( 4.0,-4.0) : the end of vector x
(-3.5, 0.0)
(-11.5, 1.0)
(-27.5, 3.0)
(-29.0, 7.5)
(-25.5, 10.0)
(-14.5, 10.0) : the end of vector y

```

### 10.3 Program Results

nag\_zgbmv (f16sbc) Example Program Results

```
      Y
(   1.000000,   2.000000)
(   3.000000,   4.000000)
(   5.000000,   6.000000)
(   7.000000,   8.000000)
(   9.000000,  10.000000)
(  11.000000,  12.000000)
```

---