

NAG C Library Function Document

nag_zhbmw (f16sdc)

1 Purpose

nag_zhbmw (f16sdc) performs matrix-vector multiplication for a complex Hermitian band matrix.

2 Specification

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

```
void nag_zhbmw (Nag_OrderType order, Nag_UploType uplo, Integer n, Integer k,
               Complex alpha, const Complex ab[], Integer pdab, const Complex x[],
               Integer incx, Complex beta, Complex y[], Integer incy, NagError *fail)
```

3 Description

nag_zhbmw (f16sdc) performs the matrix-vector operation

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

where A is an n by n complex Hermitian band matrix with k subdiagonals and k superdiagonals, x and y are n element complex vectors, and α and β are complex scalars.

4 References

The BLAS Technical Forum Standard (2001) 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 the upper or lower triangular part of A is stored.
uplo = Nag_Upper
The upper triangular part of A is stored.
uplo = Nag_Lower
The lower triangular part of A is stored.
Constraint: **uplo = Nag_Upper** or **Nag_Lower**.
- 3: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.

- 4: **k** – Integer *Input*
On entry: k , the number of subdiagonals or superdiagonals of the matrix A .
Constraint: $k \geq 0$.
- 5: **alpha** – Complex *Input*
On entry: the scalar α .
- 6: **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 symmetric 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 $j = \max(1, i - k), \dots, i$.
- 7: **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: $\mathbf{pdab} \geq \mathbf{k} + 1$.
- 8: **x**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.
On entry: the vector x .
- 9: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: $\mathbf{incx} \neq 0$.
- 10: **beta** – Complex *Input*
On entry: the scalar β .
- 11: **y**[*dim*] – Complex *Input/Output*
Note: the dimension, *dim*, of the array **y** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incy}|)$.
On entry: the vector y .
 If **beta** = 0, **y** need not be set.
On exit: the updated vector y .

- 12: **incy** – Integer *Input*
On entry: the increment in the subscripts of **y** between successive elements of **y**.
Constraint: **incy** $\neq 0$.
- 13: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **inex** = $\langle value \rangle$.
Constraint: **inex** $\neq 0$.

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

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** $\geq k + 1$.

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

None.

9 Example

To compute the matrix-vector product

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

where

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

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

$$y = \begin{pmatrix} 3.0 - 0.5i \\ -0.5 - 6.0i \\ 0.5 - 8.5i \\ 2.5 - 6.0i \\ 14.0 - 2.0i \end{pmatrix},$$

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

9.1 Program Text

```

/* nag_zhbmV (f16sdc) 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, beta;
    Integer exit_status, i, incx, incy, j, k, kd, 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_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_zhbmV (f16sdc) 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 scalar parameters */
    Vscanf(" ( %lf , %lf ) ( %lf , %lf )%*[\n] ",
           &alpha.re, &alpha.im, &beta.re, &beta.im);
    /* Read increment parameters */
    Vscanf("%ld%ld%*[\n] ", &incx, &incy);

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

    if (n > 0)
    {
        /* Allocate memory */
        if ( !(ab = NAG_ALLOC(pdab*n, Complex)) ||
             !(x = NAG_ALLOC(xlen, Complex)) ||
             !(y = NAG_ALLOC(ylen, 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 vectors x and y */

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

    /* nag_zhbmvm(f16sdc).
    * Hermitian banded matrix-vector multiply.
    *
    */
    nag_zhbmvm(order, uplo, n, kd, alpha, ab, pdab, x, incx,
               beta, y, incy, &fail);
    if (fail.code != NE_NOERROR)
    {
        Vprintf("Error from nag_zhbmvm.\n%s\n", fail.message);
    }

```

```

        exit_status = 1;
        goto END;
    }

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

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

    return exit_status;
}

```

9.2 Program Data

```

nag_zhbmV (f16sdc) Example Program Data
  5  2          :Values of n and kd
  Nag_Lower    :Value of uplo
  ( 1.0, 0.0) ( 2.0, 0.0)      : alpha, beta
  1  1          : incx, incy
  (1.0, 0.0)
  (2.0, 1.0) (2.0, 0.0)
  (3.0, 1.0) (3.0, 2.0) (3.0, 0.0)
                    (4.0, 2.0) (4.0, 3.0) (4.0, 0.0)
                    (5.0, 3.0) (5.0, 4.0) (5.0, 0.0) :End of matrix A

  (-1.0, 1.0)
  ( 2.0, 2.0)
  (-3.0,-1.0)
  ( 2.0, 3.0)
  (-1.0, 1.0)          : the end of vector x
  ( 3.0,-0.5)
  (-0.5,-6.0)
  ( 0.5,-8.5)
  ( 2.5,-6.0)
  (14.0,-2.0)          : the end of vector y

```

9.3 Program Results

```

nag_zhbmV (f16sdc) Example Program Results

```

```

  y
  ( 1.000000,  2.000000)
  ( 3.000000,  4.000000)
  ( 5.000000,  6.000000)
  ( 7.000000,  8.000000)
  ( 9.000000, 10.000000)

```
