NAG C 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

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

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 α and β are complex scalars.

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

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: **trans** – Nag_TransType *Input*

On entry: specifies the operation to be performed.

trans = **Nag_NoTrans**

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

trans = **Nag_Trans**

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

trans = **Nag_ConjTrans**

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

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: **m** ≥ 0.

4: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: **n** ≥ 0.

5: **kl** – Integer *Input*

On entry: k_l, the number of subdiagonals within the band of A.

Constraint: **kl** ≥ 0.

6: **ku** – Integer *Input*

On entry: k_u, the number of superdiagonals within the band of A.

Constraint: **ku** ≥ 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

$\max(1, \textbf{pdab} \times \textbf{n})$ when **order** = **Nag_ColMajor**;
$\max(1, \textbf{pdab} \times \textbf{m})$ when **order** = **Nag_RowMajor**.

On entry: the m by n 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 $i = 1, \ldots, m$ and $j = \max(1, i - k_l), \ldots, \min(n, i + k_u)$, depends on the **order** argument as follows:

if **order** = **Nag_ColMajor**, a_{ij} is stored as **ab**$[(j-1) \times \textbf{pdab} + \textbf{ku} + i - j]$;
if **order** = **Nag_RowMajor**, a_{ij} is stored as **ab**$[(i-1) \times \textbf{pdab} + \textbf{kl} + j - i]$.

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: **pdab** $\geq$ **kl** + **ku** + 1.

10: **x**[dim] – const Complex *Input*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, 1 + (\textbf{n} - 1)|\textbf{incx}|)$ when **trans** = **Nag_NoTrans**;
$\max(1, 1 + (\textbf{m} - 1)|\textbf{incx}|)$ when **trans** = **Nag_Trans** or **Nag_ConjTrans**.

On entry: the 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: **beta** – Complex *Input*

On entry: the scalar β.

13: **y**[*dim*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **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*.

If **beta** = 0, **y** need not be set.

On exit: the updated vector *y*.

14: **incy** – Integer *Input*

On entry: the increment in the subscripts of **y** between successive elements of *y*.

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

15: **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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **incx** = ⟨*value*⟩.
Constraint: **incx** $\neq 0$.

On entry, **incy** = ⟨*value*⟩.
Constraint: **incy** $\neq 0$.

On entry, **kl** = ⟨*value*⟩.
Constraint: **kl** ≥ 0.

On entry, **ku** = ⟨*value*⟩.
Constraint: **ku** ≥ 0.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_3

On entry, **pdab** = ⟨*value*⟩, **kl** = ⟨*value*⟩, **ku** = ⟨*value*⟩.
Constraint: **pdab** $\geq$ **kl** + **ku** + 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

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

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

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

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

9.1 Program Text

```
/* nag_zgbmv (f16sbc) 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 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);

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

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  /* Read the problem dimensions */
  Vscanf("%ld%ld%ld%ld%*[^\n] ",
         &m, &n, &kl, &ku);
  /* Read the transpose parameter */
  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 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 = 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)))
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      Vprintf("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)
        Vscanf(" ( %lf , %lf )", &AB(i,j).re, &AB(i,j).im);
```

```
      Vscanf("%*[^\n] ");
    }
  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_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)
    {
      Vprintf("Error from nag_zgbmv.\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_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
```

9.3 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)
```