

NAG Library Function Document

nag_dgbmv (f16pbc)

1 Purpose

nag_dgbmv (f16pbc) performs matrix-vector multiplication for a real band matrix.

2 Specification

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

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

3 Description

nag_dgbmv (f16pbc) performs one of the matrix-vector operations

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

where A is an m by n real band matrix with k_l subdiagonals and k_u superdiagonals, x and y are real vectors, and α and β are real 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 or Nag_ConjTrans

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

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** – double *Input*
On entry: the scalar α .
- 8: **ab**[*dim*] – const double *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 double *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** – double *Input*
On entry: the scalar β .
- 13: **y[dim]** – double *Input/Output*
Note: the dimension, 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** ≥ 0 .

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

Constraint: **m** ≥ 0 .

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

Constraint: **n** ≥ 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_dgbmv (f16pbc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

A vector y , of length 6, is updated using $y \leftarrow 2y + Ax$, where A is a 6 by 4 banded matrix with two subdiagonals and one superdiagonal, and x is a vector of length 4.

10.1 Program Text

```
/* nag_dgbmv (f16pbc) 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 */
    double alpha, beta;
    Integer ab_size, exit_status, i, incx, incy, j, kl, ku;
```

```

Integer m, n, pdab, xlen, ylen;

/* Arrays */
double *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_dgbmv (f16pbc) 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%*[\n] ", &alpha, &beta);
#else
    scanf("%lf %lf%*[\n] ", &alpha, &beta);
#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, double)) ||
        !(x = NAG_ALLOC(xlen, double)) || !(y = NAG_ALLOC(ylen, double)))
    {
        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", &AB(i, j));
#else
        scanf("%lf", &AB(i, j));
#endif
}
for (i = 1; i <= xlen; ++i)
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
}
for (i = 1; i <= ylen; ++i)
#ifdef _WIN32
    scanf_s("%lf%*[^\\n] ", &x[i - 1]);
#else
    scanf("%lf%*[^\\n] ", &x[i - 1]);
#endif
for (i = 1; i <= ylen; ++i)
#ifdef _WIN32
    scanf_s("%lf%*[^\\n] ", &y[i - 1]);
#else
    scanf("%lf%*[^\\n] ", &y[i - 1]);
#endif

/* nag_dgbmv (f16pbc).
 * real valued band matrix-vector multiply.
 */
nag_dgbmv(order, trans, m, n, kl, ku, alpha, ab, pdab, x,
          incx, beta, y, incy, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dgbmv.\\n%s\\n", fail.message);
    exit_status = 1;
    goto END;
}

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

END:
    NAG_FREE(ab);

```

```

    NAG_FREE(x);
    NAG_FREE(y);

    return exit_status;
}

```

10.2 Program Data

```

nag_dgbmv (f16pbc) Example Program Data
  6 4 2 1      :Values of m, n, kl, ku
  Nag_NoTrans  : trans
  1.0 2.0      : alpha, beta
  1 1          : incx, incy
  1.0 1.0
  2.0 2.0 2.0
  3.0 3.0 3.0 3.0
      4.0 4.0 4.0
      5.0 5.0
      6.0      : the end of matrix A

  1.0
  2.0
  3.0
  4.0          : the end of vector x
 -0.5
 -4.5
 -13.0
 -15.5
 -14.5
 -8.5         : the end of vector y

```

10.3 Program Results

```

nag_dgbmv (f16pbc) Example Program Results

```

Updated vector y:

```

  2.000000
  3.000000
  4.000000
  5.000000
  6.000000
  7.000000

```
