NAG Library Function Document

nag_dsbmv (f16pdc)

1 Purpose

nag_dsbmv (f16pdc) performs matrix-vector multiplication for a real symmetric band matrix.

2 Specification

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

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

3 Description

nag_dsbmv (f16pdc) performs the matrix-vector operation

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

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

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 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** ≥ 0.

4: **k** – Integer *Input*

On entry: k, the number of subdiagonals or superdiagonals of the matrix A.

Constraint: $\mathbf{k} \geq 0$.

5: **alpha** – double *Input*

On entry: the scalar α.

6: **ab**[dim] – const double *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 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 $\mathbf{ab}[k + i - j + (j - 1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = \max(1, j - k), \ldots, j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[i - j + (j - 1) \times \mathbf{pdab}]$, for $j = 1, \ldots, n$ and $i = j, \ldots, \min(n, j + k)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ab}[j - i + (i - 1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = i, \ldots, \min(n, i + k)$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ab}[k + j - i + (i - 1) \times \mathbf{pdab}]$, for $i = 1, \ldots, n$ and $j = \max(1, i - k), \ldots, i$.

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 double *Input*

Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.

On entry: the n-element vector x.

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

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

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

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** – double *Input*

On entry: the scalar β.

11: **y**[dim] – double *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. See **x** for details of storage.

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 2.7 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 ⟨*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, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.

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

NE_INT_2

On entry, **pdab** = ⟨*value*⟩, **k** = ⟨*value*⟩.
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_dsbmv (f16pdc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example computes the matrix-vector product

$$y = \alpha A x + \beta y$$

where

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

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

$$y = \begin{pmatrix} 10.0 \\ 1.5 \\ 9.5 \\ 8.5 \\ 24.0 \end{pmatrix},$$

$$\alpha = 1.5 \quad \text{and} \quad \beta = 1.0.$$

10.1 Program Text

```
/* nag_dsbmv (f16pdc) 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 exit_status, i, incx, incy, j, k, kd, 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_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_dsbmv (f16pdc) 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 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 = 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, double)) ||
        !(x = NAG_ALLOC(xlen, double)) || !(y = NAG_ALLOC(ylen, double)))
    {
      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", &AB_UPPER(i, j));
#else
        scanf("%lf", &AB_UPPER(i, j));
#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", &AB_LOWER(i, j));
#else
        scanf("%lf", &AB_LOWER(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  /* Input vectors x and y */

  for (i = 1; i <= xlen; ++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_dsbmv (f16pdc).
   * Symmetric banded matrix-vector multiply.
   *
   */
  nag_dsbmv(order, uplo, n, kd, alpha, ab, pdab, x, incx,
            beta, y, incy, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dsbmv.\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\n", y[i - 1]);
  }

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

  return exit_status;
}
```

10.2 Program Data

```
nag_dsbmv (f16pdc) Example Program Data
  5  2                     :Values of n and kd
  Nag_Lower                :Value of uplo
  1.5 1.0              : alpha, beta
  1 1                  : incx, incy
  1.0
  2.0  2.0
  3.0  3.0  3.0
       4.0  4.0  4.0
            5.0  5.0  5.0   :End of matrix A
   -1.0
    2.0
   -3.0
    2.0
   -1.0                : the end of vector x
   10.0
    1.5
    9.5
    8.5
   24.0                : the end of vector y
```

10.3 Program Results

```
nag_dsbmv (f16pdc) Example Program Results

  y
   1.000000
   3.000000
   5.000000
   7.000000
   9.000000
```