

# NAG Library Function Document

## nag\_zspmv (f16tcc)

### 1 Purpose

nag\_zspmv (f16tcc) performs matrix-vector multiplication for a complex symmetric matrix stored in packed form.

### 2 Specification

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

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

### 3 Description

nag\_zspmv (f16tcc) performs the matrix-vector operation

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

where  $A$  is an  $n$  by  $n$  complex symmetric matrix stored in packed form,  $x$  and  $y$  are  $n$ -element complex vectors, and  $\alpha$  and  $\beta$  are complex 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:*  $n \geq 0$ .

- 4: **alpha** – Complex *Input*  
*On entry:* the scalar  $\alpha$ .
- 5: **ap**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **ap** must be at least  $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$ .  
*On entry:* the  $n$  by  $n$  symmetric matrix  $A$ , packed by rows or columns.  
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 **ap**[( $j - 1$ )  $\times$   $j/2 + i - 1$ ], for  $i \leq j$ ;  
if **order** = Nag\_ColMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap**[( $2n - j$ )  $\times$  ( $j - 1$ )/2 +  $i - 1$ ], for  $i \geq j$ ;  
if **order** = Nag\_RowMajor and **uplo** = Nag\_Upper,  
 $A_{ij}$  is stored in **ap**[( $2n - i$ )  $\times$  ( $i - 1$ )/2 +  $j - 1$ ], for  $i \leq j$ ;  
if **order** = Nag\_RowMajor and **uplo** = Nag\_Lower,  
 $A_{ij}$  is stored in **ap**[( $i - 1$ )  $\times$   $i/2 + j - 1$ ], for  $i \geq j$ .
- 6: **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  $n$ -element vector  $x$ .  
If **incx** > 0,  $x_i$  must be stored in **x**[( $i - 1$ )  $\times$  **incx**], for  $i = 1, 2, \dots, \mathbf{n}$ .  
If **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 **n** = 0, **x** is not referenced and may be **NULL**.
- 7: **incx** – Integer *Input*  
*On entry:* the increment in the subscripts of **x** between successive elements of  $x$ .  
**Constraint:** **incx**  $\neq$  0.
- 8: **beta** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 9: **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  $y$ . See **x** for details of storage.  
If **beta** = 0, **y** need not be set.  
*On exit:* the updated vector  $y$ .
- 10: **incy** – Integer *Input*  
*On entry:* the increment in the subscripts of **y** between successive elements of  $y$ .  
**Constraint:** **incy**  $\neq$  0.
- 11: **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  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry,  $\mathbf{incx} = \langle value \rangle$ .

Constraint:  $\mathbf{incx} \neq 0$ .

On entry,  $\mathbf{incy} = \langle value \rangle$ .

Constraint:  $\mathbf{incy} \neq 0$ .

On entry,  $\mathbf{n} = \langle value \rangle$ .

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

### 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\_zspmv (f16tcc) 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

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

$$x = \begin{pmatrix} 1.0 + 0.0i \\ 0.0 - 1.0i \\ -1.0 + 0.0i \\ 0.0 + 1.0i \end{pmatrix},$$

$$y = \begin{pmatrix} 10.0 + 4.0i \\ 10.0 + 8.0i \\ 10.0 + 16.0i \\ 14.0 + 24.0i \end{pmatrix},$$

$$\alpha = 1.0 + 1.0i \quad \text{and} \quad \beta = 0.5 + 0.0i.$$

## 10.1 Program Text

```

/* nag_zspmv (f16tcc) 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 aplen, exit_status, i, incx, incy, j, n, xlen, ylen;

    /* Arrays */
    Complex *ap = 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 A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
    order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_zspmv (f16tcc) 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 "%*[\n] ", &n);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &n);
#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 ) ( %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

    aplen = n * (n + 1) / 2;
    xlen = MAX(1, 1 + (n - 1) * ABS(incx));
    ylen = MAX(1, 1 + (n - 1) * ABS(incy));

    if (n > 0) {
        /* Allocate memory */
        if (!(ap = NAG_ALLOC(aplen, Complex)) ||
            !(x = NAG_ALLOC(xlen, Complex)) || !(y = NAG_ALLOC(ylen, Complex)))
        {
            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 */
    if (uplo == Nag_Upper) {
        for (i = 1; i <= n; ++i) {
            for (j = i; j <= n; ++j)
#ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#else
                scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#endif
        }
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    }
    else {
        for (i = 1; i <= n; ++i) {
            for (j = 1; j <= i; ++j)
#ifdef _WIN32
                scanf_s(" ( %lf , %lf ) ", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
#else

```

```

        scanf(" ( %lf , %lf ) ", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
#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 , %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_zspmv (f16tcc).
 * Complex symmetric packed storage matrix-vector multiply.
 */
nag_zspmv(order, uplo, n, alpha, ap, x, incx, beta, y, incy, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zspmv.\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(ap);
    NAG_FREE(x);
    NAG_FREE(y);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_zspmv (f16tcc) Example Program Data
4                               :Value of n
Nag_Lower                       :Value of uplo
( 1.0, 1.0) ( 0.5, 0.0)         :Values of alpha, beta
1 1                               :Values of incx, incy
( 1.0, 1.0)
( 2.0, 1.0) ( 2.0, 2.0)
( 3.0, 1.0) ( 3.0, 2.0) ( 3.0, 3.0)
( 4.0, 1.0) ( 4.0, 2.0) ( 4.0, 3.0) ( 4.0, 4.0) :End of matrix A
( 1.0, 0.0)
( 0.0,-1.0)
(-1.0, 0.0)
( 0.0, 1.0)                       :End of vector x
(10.0, 4.0)
(10.0, 8.0)
(10.0,16.0)
(14.0,24.0)                       : the end of vector y

```

### **10.3 Program Results**

nag\_zspmv (f16tcc) Example Program Results

```
      Y  
(  1.000000,  2.000000)  
(  3.000000,  4.000000)  
(  5.000000,  6.000000)  
(  7.000000,  8.000000)
```

---