

# NAG C Library Function Document

## nag\_zsymv (f16tac)

### 1 Purpose

nag\_zsymv (f16tac) performs matrix-vector multiplication for a complex symmetric matrix.

### 2 Specification

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

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

### 3 Description

nag\_zsymv (f16tac) performs the matrix-vector operation

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

where  $A$  is an  $n$  by  $n$  complex symmetric matrix,  $x$  and  $y$  are  $n$  element complex vectors, and  $\alpha$  and  $\beta$  are complex scalars.

### 4 References

The BLAS Technical Forum Standard (2001) [www.netlib.org/blas/blast-forum](http://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: **alpha** – Complex *Input*  
*On entry:* the scalar  $\alpha$ .
- 5: **a**[*dim*] – const Complex *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times \mathbf{n})$ .  
 If **order** = **Nag\_ColMajor**, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*j* – 1) × **pda** + *i* – 1].  
 If **order** = **Nag\_RowMajor**, the (*i*,*j*)th element of the matrix *A* is stored in **a**[(*i* – 1) × **pda** + *j* – 1].  
*On entry:* the *n* by *n* symmetric matrix *A*.  
 If **uplo** = **Nag\_Upper**, the upper triangle of *A* must be stored and the elements of the array below the diagonal are not referenced.  
 If **uplo** = **Nag\_Lower**, the lower triangle of *A* must be stored and the elements of the array above the diagonal are not referenced.
- 6: **pda** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in the array **a**.  
**Constraint:** **pda** ≥  $\max(1, \mathbf{n})$ .
- 7: **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*.
- 8: **incx** – Integer *Input*  
*On entry:* the increment in the subscripts of **x** between successive elements of *x*.  
**Constraint:** **incx** ≠ 0.
- 9: **beta** – Complex *Input*  
*On entry:* the scalar  $\beta$ .
- 10: **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*.
- 11: **incy** – Integer *Input*  
*On entry:* the increment in the subscripts of **y** between successive elements of *y*.  
**Constraint:** **incy** ≠ 0.
- 12: **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, **incx** =  $\langle value \rangle$ .

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

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

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

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

Constraint: **n**  $\geq 0$ .

**NE\_INT\_2**

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

Constraint: **pda**  $\geq \max(1, \mathbf{n})$ .

**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

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

$$x = \begin{pmatrix} -1.0 + 0.0i \\ 0.0 + 2.0i \\ -3.0 + 1.0i \end{pmatrix},$$

$$y = \begin{pmatrix} 6.0 + 4.5i \\ 8.5 + 4.5i \\ 12.0 + 5.5i \end{pmatrix},$$

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

**9.1 Program Text**

```

/* nag_zsymv (f16tac) 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, n, pda, xlen, ylen;

    /* Arrays */
    Complex *a=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(I,J) a[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I,J) a[(I-1)*pda + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);
    Vprintf( "nag_zsymv (f16tac) Example Program Results\n\n");

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

    /* Read the problem dimension */
    Vscanf("%ld%*[^\\n] ", &n);

    /* 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);

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

    if (n > 0)
    {
        /* Allocate memory */
        if ( !(a = NAG_ALLOC(n*pda, 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;
    }
}

```

```

/* Input the matrix A and vectors x and y */
if (uplo == Nag_Upper)
{
  for (i = 1; i <= n; ++i)
  {
    for (j = i; j <= n; ++j)
      Vscanf(" ( %lf , %lf ) ", &A(i,j).re, &A(i,j).im);
  }
  Vscanf("%*[\n] ");
}
else
{
  for (i = 1; i <= n; ++i)
  {
    for (j = 1; j <= i; ++j)
      Vscanf(" ( %lf , %lf ) ", &A(i,j).re, &A(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_zsymv(f16tac).
 * Complex symmetric matrix-vector multiply.
 */
nag_zsymv(order, uplo, n, alpha, a, pda, x, incx, beta,
          y, incy, &fail );
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from nag_zsymv.\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 (a) NAG_FREE(a);
if (x) NAG_FREE(x);
if (y) NAG_FREE(y);

return exit_status;
}

```

## 9.2 Program Data

```

nag_zsymv (f16tac) Example Program Data
3                               : n the dimension of matrix A
Nag_Upper                       : uplo
(1.0, 0.0) ( 2.0, 0.0) : alpha, beta
1 1                               : incx, incy
(1.0, 1.0) ( 1.0, 2.0) ( 1.0, 3.0)
      ( 2.0, 2.0) ( 2.0, 3.0)
      ( 3.0, 3.0) : the end of matrix A

(-1.0, 0.0)
( 0.0, 2.0)
(-3.0, 1.0)                   : the end of vector x
( 6.0, 4.5)
( 8.5, 4.5)
(12.0, 5.5)                   : the end of vector y

```

### 9.3 Program Results

nag\_zsymv (f16tac) Example Program Results

```
      y  
(  1.000000,  2.000000)  
(  3.000000,  4.000000)  
(  5.000000,  6.000000)
```

---