

NAG 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 α and β 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 α .
- 5: **a**[*dim*] – const Complex *Input*
Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.
On entry: the n by n symmetric matrix A .
If **order** = Nag_ColMajor, A_{ij} is stored in **a**[($j - 1$) \times **pda** + $i - 1$].
If **order** = Nag_RowMajor, A_{ij} is stored in **a**[($i - 1$) \times **pda** + $j - 1$].
If **uplo** = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.
If **uplo** = Nag_Lower, the lower triangular part 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 row or column elements (depending on the value of **order**) of the matrix A in the array **a**.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.
- 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 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**.
- 8: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: **incx** $\neq 0$.
- 9: **beta** – Complex *Input*
On entry: the scalar β .
- 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 . See **x** for details of storage.
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** $\neq 0$.
- 12: **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_INT_2

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

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

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_zsymv (f16tac) 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

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

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

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

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

10.1 Program Text

```

/* nag_zsymv (f16tac) 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 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);

    printf("nag_zsymv (f16tac) 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

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)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
}
else {
printf("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)
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
#else
scanf(" ( %lf , %lf ) ", &A(i, j).re, &A(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(i, j).re, &A(i, j).im);
#else
scanf(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
#endif
}
}
}

```

```

#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    }
    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_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) {
    printf("Error from nag_zsymv.\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(a);
    NAG_FREE(x);
    NAG_FREE(y);

    return exit_status;
}

```

10.2 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

```

10.3 Program Results

nag_zsymv (f16tac) Example Program Results

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