

NAG Library Function Document

nag_ztrmv (f16sfc)

1 Purpose

nag_ztrmv (f16sfc) performs matrix-vector multiplication for a complex triangular matrix.

2 Specification

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

void nag_ztrmv (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
               Nag_DiagType diag, Integer n, Complex alpha, const Complex a[],
               Integer pda, Complex x[], Integer incx, NagError *fail)
```

3 Description

nag_ztrmv (f16sfc) performs one of the matrix-vector operations

$$x \leftarrow \alpha Ax, \quad x \leftarrow \alpha A^T x \quad \text{or} \quad x \leftarrow \alpha A^H x,$$

where A is an n by n complex triangular matrix, and x is an n -element complex vector and α is a complex scalar.

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 A is upper or lower triangular.
uplo = Nag_Upper
 A is upper triangular.
uplo = Nag_Lower
 A is lower triangular.
Constraint: **uplo** = Nag_Upper or Nag_Lower.
- 3: **trans** – Nag_TransType *Input*
On entry: specifies the operation to be performed.
trans = Nag_NoTrans
 $x \leftarrow \alpha Ax$.

- trans** = Nag_Trans
 $x \leftarrow \alpha A^T x.$
- trans** = Nag_ConjTrans
 $x \leftarrow \alpha A^H x.$
- Constraint:* **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.
- 4: **diag** – Nag_DiagType *Input*
On entry: specifies whether A has nonunit or unit diagonal elements.
diag = Nag_NonUnitDiag
 The diagonal elements are stored explicitly.
diag = Nag_UnitDiag
 The diagonal elements are assumed to be 1 and are not referenced.
Constraint: **diag** = Nag_NonUnitDiag or Nag_UnitDiag.
- 5: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.
- 6: **alpha** – Complex *Input*
On entry: the scalar α .
- 7: **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 triangular 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.
 If **diag** = Nag_UnitDiag, the diagonal elements of A are assumed to be 1, and are not referenced.
- 8: **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** $\geq \max(1, \mathbf{n})$.
- 9: **x**[*dim*] – Complex *Input/Output*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$.
On entry: the right-hand side vector b .
On exit: the solution vector x .
- 10: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: **incx** $\neq 0$.

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, **incx** = $\langle value \rangle$.

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

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_ztrmv (f16sfc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example computes the matrix-vector product

$$y = \alpha Ax$$

where

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

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

and

$$\alpha = 1.0 + 0.0i.$$

10.1 Program Text

```

/* nag_ztrmv (f16sfc) 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;
    Integer exit_status, i, incx, j, n, pda, xlen;

    /* Arrays */
    Complex *a = 0, *x = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail;
    Nag_DiagType diag;
    Nag_OrderType order;
    Nag_TransType trans;
    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_ztrmv (f16sfc) 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 trans */
#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 diag */
#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
     */
    diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);
    /* Read scalar parameters */
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
#else
    scanf(" ( %lf , %lf )%*[\n] ", &alpha.re, &alpha.im);
#endif
    /* Read increment parameters */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[\n] ", &incx);
#else
    scanf("%" NAG_IFMT "%*[\n] ", &incx);
#endif

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

    if (n > 0) {
        /* Allocate memory */
        if (!(a = NAG_ALLOC(n * pda, Complex)) || !(x = NAG_ALLOC(xlen, 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(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
        }
#ifdef _WIN32
        scanf_s("%*[^\\n] ");
#else
        scanf("%*[^\\n] ");
#endif
    }

    /* Input vector x */
    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

    /* nag_ztrmv (f16sfc).
     * Complex triangular matrix-vector multiply.
     */
    nag_ztrmv(order, uplo, trans, diag, n, alpha, a, pda, x, incx, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_ztrmv (f16sfc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Print output vector x */
    printf("%s\n", " x");
    for (i = 1; i <= xlen; ++i)
        printf("(%11f,%11f)\n", x[i - 1].re, x[i - 1].im);

END:
    NAG_FREE(a);
    NAG_FREE(x);

    return exit_status;
}

```

10.2 Program Data

```

nag_ztrmv (f16sfc) Example Program Data
4                               :Value of n
Nag_Lower                       :Value of uplo
Nag_NoTrans                      :Value of trans
Nag_NonUnitDiag                 :Value of diag
( 1.0, 0.0)                     :Value of alpha
1                               :Value of incx
( 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, 1.0)
( 2.0,-2.0)
(-3.0, 2.0)
(-2.0, 1.0) :End of vector x
```

10.3 Program Results

nag_ztrmv (f16sfc) Example Program Results

```
 x
( -2.000000,  0.000000)
(  5.000000,  1.000000)
( -9.000000, -3.000000)
( -23.000000, -6.000000)
```
