

# NAG Library Function Document

## nag\_dtrmv (f16pfc)

### 1 Purpose

nag\_dtrmv (f16pfc) performs matrix-vector multiplication for a real triangular matrix.

### 2 Specification

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

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

### 3 Description

nag\_dtrmv (f16pfc) performs one of the matrix-vector operations

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

where  $A$  is an  $n$  by  $n$  real triangular matrix,  $x$  is an  $n$ -element real vector and  $\alpha$  is a real 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 or Nag\_ConjTrans

$$x \leftarrow \alpha A^T x.$$

*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** – double

*Input*

*On entry:* the scalar  $\alpha$ .

7: **a**[ $dim$ ] – const double

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

9: **x**[ $dim$ ] – double

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

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\_dtrmv (f16pfc) is not threaded in any implementation.

## 9 Further Comments

None.

## 10 Example

This example computes the matrix-vector product

$$y = \alpha Ax$$

where

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

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

and

$$\alpha = 1.5.$$

## 10.1 Program Text

```

/* nag_dtrmv (f16pfc) 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;
    Integer exit_status, i, incx, j, n, pda, xlen;

    /* Arrays */
    double *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_dtrmv (f16pfc) 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%*[\n] ", &alpha);
#else
    scanf("%lf%*[\n] ", &alpha);
#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, double)) || !(x = NAG_ALLOC(xlen, 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 */
    if (uplo == Nag_Upper) {
        for (i = 1; i <= n; ++i) {
            for (j = i; j <= n; ++j)

```

```

#ifdef _WIN32
    scanf_s("%lf", &A(i, j));
#else
    scanf("%lf", &A(i, j));
#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", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
}

/* Input vector x */
for (i = 1; i <= xlen; ++i)
#ifdef _WIN32
    scanf_s("%lf%*[^\\n] ", &x[i - 1]);
#else
    scanf("%lf%*[^\\n] ", &x[i - 1]);
#endif

/* nag_dtrmv (f16pfc).
 * Triangular matrix-vector multiply.
 */
nag_dtrmv(order, uplo, trans, diag, n, alpha, a, pda, x, incx, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrmv (f16pfc).\\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\\n", x[i - 1]);
}

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

    return exit_status;
}

```

## 10.2 Program Data

nag\_dtrmv (f16pfc) Example Program Data

4	:Value of n
Nag_Lower	:Value of uplo
Nag_NoTrans	:Value of trans
Nag_NonUnitDiag	:Value of diag
1.5	:Value of alpha
1	:Value of incx
1.0	

```
2.0    2.0
3.0    3.0    3.0
4.0    4.0    4.0    4.0    :End of matrix A
-1.0
 2.0
-3.0
 1.0    :End of vector x
```

### 10.3 Program Results

nag\_dtrmv (f16pfc) Example Program Results

```
x
-1.500000
 3.000000
-9.000000
-6.000000
```

---