

# NAG Library Function Document

## nag\_dtrsm (f16yjc)

### 1 Purpose

nag\_dtrsm (f16yjc) solves a system of equations given as a real triangular matrix with multiple right-hand sides.

### 2 Specification

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

void nag_dtrsm (Nag_OrderType order, Nag_SideType side, Nag_UploType uplo,
               Nag_TransType trans, Nag_DiagType diag, Integer m, Integer n,
               double alpha, const double a[], Integer pda, double b[], Integer pdb,
               NagError *fail)
```

### 3 Description

nag\_dtrsm (f16yjc) performs one of the matrix-matrix operations

$$\begin{aligned} B &\leftarrow \alpha A^{-1}B, & B &\leftarrow \alpha A^{-T}B, \\ B &\leftarrow \alpha BA^{-1} & \text{or} & B &\leftarrow \alpha BA^{-T}, \end{aligned}$$

where  $A$  is a real triangular matrix,  $B$  is an  $m$  by  $n$  real matrix, and  $\alpha$  is a real scalar.  $A^{-T}$  denotes  $(A^T)^{-1}$  or equivalently  $(A^{-1})^T$ .

### 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: **side** – Nag\_SideType *Input*

*On entry:* specifies whether  $B$  is operated on from the left or the right.

**side** = Nag\_LeftSide

$B$  is pre-multiplied from the left.

**side** = Nag\_RightSide

$B$  is post-multiplied from the right.

*Constraint:* **side** = Nag\_LeftSide or Nag\_RightSide.

- 3:     **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.
- 4:     **trans** – Nag\_TransType *Input*  
*On entry:* specifies the operation to be performed.  
**trans** = Nag\_Trans or Nag\_ConjTrans and **side** = Nag\_LeftSide  
 $B \leftarrow \alpha A^{-T} B$ .  
**trans** = Nag\_NoTrans and **side** = Nag\_LeftSide  
 $B \leftarrow \alpha A^{-1} B$ .  
**trans** = Nag\_Trans or Nag\_ConjTrans and **side** = Nag\_RightSide  
 $B \leftarrow \alpha B A^{-T}$ .  
**trans** = Nag\_NoTrans and **side** = Nag\_RightSide  
 $B \leftarrow \alpha B A^{-1}$ .  
*Constraint:* **trans** = Nag\_NoTrans, Nag\_Trans or Nag\_ConjTrans.
- 5:     **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.
- 6:     **m** – Integer *Input*  
*On entry:*  $m$ , the number of rows of the matrix  $B$ ; the order of  $A$  if **side** = Nag\_LeftSide.  
*Constraint:* **m**  $\geq 0$ .
- 7:     **n** – Integer *Input*  
*On entry:*  $n$ , the number of columns of the matrix  $B$ ; the order of  $A$  if **side** = Nag\_RightSide.  
*Constraint:* **n**  $\geq 0$ .
- 8:     **alpha** – double *Input*  
*On entry:* the scalar  $\alpha$ .
- 9:     **a**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **a** must be at least  
 $\max(1, \mathbf{pda} \times \mathbf{m})$  when **side** = Nag\_LeftSide;  
 $\max(1, \mathbf{pda} \times \mathbf{n})$  when **side** = Nag\_RightSide.  
*On entry:* the triangular matrix  $A$ ;  $A$  is  $m$  by  $m$  if **side** = Nag\_LeftSide, or  $n$  by  $n$  if **side** = Nag\_RightSide.  
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  $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ .

If **uplo** = Nag\_Upper,  $A$  is upper triangular and the elements of the array corresponding to the lower triangular part of  $A$  are not referenced.

If **uplo** = Nag\_Lower,  $A$  is lower triangular and the elements of the array corresponding to the upper triangular part of  $A$  are not referenced.

If **diag** = Nag\_UnitDiag, the diagonal elements of  $A$  are assumed to be 1, and are not referenced.

10: **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**.

*Constraints:*

if **side** = Nag\_LeftSide,  $\mathbf{pda} \geq \max(1, \mathbf{m})$ ;

if **side** = Nag\_RightSide,  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .

11: **b**[*dim*] – double *Input/Output*

**Note:** the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{n})$  when **order** = Nag\_ColMajor;

$\max(1, \mathbf{m} \times \mathbf{pdb})$  when **order** = Nag\_RowMajor.

If **order** = Nag\_ColMajor,  $B_{ij}$  is stored in  $\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ .

If **order** = Nag\_RowMajor,  $B_{ij}$  is stored in  $\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$ .

*On entry:* the  $m$  by  $n$  matrix  $B$ .

If **alpha** = 0, **b** need not be set.

*On exit:* the updated matrix  $B$ .

12: **pdb** – Integer *Input*

*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array **b**.

*Constraints:*

if **order** = Nag\_ColMajor,  $\mathbf{pdb} \geq \max(1, \mathbf{m})$ ;

if **order** = Nag\_RowMajor,  $\mathbf{pdb} \geq \max(1, \mathbf{n})$ .

13: **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 *value* had an illegal value.

**NE\_ENUM\_INT\_2**

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

Constraint: if **side** = Nag\_LeftSide, **pda**  $\geq \max(1, \mathbf{m})$ .

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

Constraint: if **side** = Nag\_RightSide, **pda**  $\geq \max(1, \mathbf{n})$ .

**NE\_INT**

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

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

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

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

**NE\_INT\_2**

On entry, **pdb** =  $\langle value \rangle$ , **m** =  $\langle value \rangle$ .

Constraint: **pdb**  $\geq \max(1, \mathbf{m})$ .

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

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

**NE\_INTERNAL\_ERROR**

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

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\_dtrsm (f16yjc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

**9 Further Comments**

No test for singularity or near-singularity of  $A$  is included in nag\_dtrsm (f16yjc). Such tests must be performed before calling this function.

## 10 Example

Premultiply real 4 by 2 matrix  $B$  by inverse of lower triangular 4 by 4 matrix  $A$ ,  $B \leftarrow A^{-1}B$  (or solve  $AX = B$  and return result in  $B$ ), where

$$A = \begin{pmatrix} 4.30 & & & \\ -3.96 & -4.87 & & \\ 0.40 & 0.31 & -8.02 & \\ -0.27 & 0.07 & -5.95 & 0.12 \end{pmatrix}$$

and

$$B = \begin{pmatrix} -12.90 & -21.50 \\ 16.75 & 14.93 \\ -17.55 & 6.33 \\ -11.04 & 8.09 \end{pmatrix}.$$

### 10.1 Program Text

```
/* nag_dtrsm (f16yjc) 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>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double alpha;
    Integer exit_status, i, j, m, n, pda, pdb;

    /* Arrays */
    double *a = 0, *b = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail;
    Nag_SideType side;
    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]
#define B(I, J) b[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_dtrsm (f16yjc) Example Program Results\n\n");

    /* Skip heading in data file */
```

```

#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
/* Read the problem dimensions */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\\n] ", &m, &n);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\\n] ", &m, &n);
#endif

#ifdef NAG_COLUMN_MAJOR
    pdb = m;
#else
    pdb = n;
#endif

/* Read side */
#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
 */
side = (Nag_SideType) nag_enum_name_to_value(nag_enum_arg);
/* 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), see above. */
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), see above. */
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), see above. */
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

if (side == Nag_LeftSide) {
    pda = m;
}
else {
    pda = n;
}

if (n > 0) {
    /* Allocate memory */
    if (!(a = NAG_ALLOC(pda * pda, double)) ||
        !(b = NAG_ALLOC(n * m, 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 <= pda; ++i) {
        for (j = i; j <= pda; ++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 <= pda; ++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 matrix B */
for (i = 1; i <= m; ++i) {
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &B(i, j));
#else
        scanf("%lf", &B(i, j));
#endif
}

/* nag_dtrsm (f16yjc).
 * Multiply matrix by inverse of Triangular matrix.
 */
nag_dtrsm(order, side, uplo, trans, diag, m, n, alpha, a, pda,
          b, pdb, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dtrsm (f16yjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the updated matrix B */
/* nag_gen_real_mat_print (x04cac).
 * Print real general matrix (easy-to-use)
 */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,

```

```

        n, b, pdb, "Updated matrix B", 0, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
    NAG_FREE(a);
    NAG_FREE(b);

    return exit_status;
}

```

## 10.2 Program Data

nag\_dtrsm (f16yjc) Example Program Data

4	2					:Values of m and n
Nag_LeftSide						:Value of side
Nag_Lower						:Value of uplo
Nag_NoTrans						:Value of trans
Nag_NonUnitDiag						:Value of diag
1.0						:Value of alpha
4.30						
-3.96	-4.87					
0.40	0.31	-8.02				
-0.27	0.07	-5.95	0.12			:End of matrix A
-12.90	-21.50					
16.75	14.93					
-17.55	6.33					
-11.04	8.09					:End of matrix B

## 10.3 Program Results

nag\_dtrsm (f16yjc) Example Program Results

Updated matrix B

	1	2
1	-3.0000	-5.0000
2	-1.0000	1.0000
3	2.0000	-1.0000
4	1.0000	6.0000

---