NAG Library Function Document

nag_dtrmm (f16yfc)

1 Purpose

nag_dtrmm (f16yfc) performs matrix-matrix multiplication for a real triangular matrix.

2 Specification

```
#include <nag.h>
#include <nagf16.h>
void nag_dtrmm (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_dtrmm (f16yfc) performs one of the matrix-matrix operations

$$\begin{array}{lll} B \leftarrow \alpha AB, & & B \leftarrow \alpha A^{\mathrm{T}}B, \\ B \leftarrow \alpha BA & \text{or} & B \leftarrow \alpha BA^{\mathrm{T}}, \end{array}$$

where B is an m by n real matrix, A is a real triangular matrix, and α is a real scalar.

4 References

The BLAS Technical Forum Standard (2001) 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 3.2.1.3 in the Essential Introduction 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 whether the operation involves A or A^{T}.

trans = Nag_NoTrans
It involves A.

trans = Nag_Trans or Nag_ConjTrans
It involves A^{T}.

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: $\mathbf{m} \geq 0$.

7: **n** – Integer *Input*

On entry: n, the number of columns of the matrix B; the order of A if **side** = Nag_RightSide.

Constraint: $\mathbf{n} \geq 0$.

8: **alpha** – double *Input*

On entry: the scalar α.

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 $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

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, **pda** $\geq \max(1, \mathbf{m})$;
if **side** = Nag_RightSide, **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 **b**$[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** = Nag_RowMajor, B_{ij} is stored in **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, **pdb** $\geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, **pdb** $\geq \max(1, \mathbf{n})$.

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ENUM_INT_2

On entry, **side** = ⟨*value*⟩, **m** = ⟨*value*⟩, **pda** = ⟨*value*⟩.
Constraint: if **side** = Nag_LeftSide, **pda** $\geq \max(1, \mathbf{m})$.

On entry, **side** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pda** = ⟨*value*⟩.
Constraint: if **side** = Nag_RightSide, **pda** $\geq \max(1, \mathbf{n})$.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 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.

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

Premultiply real 4 by 2 matrix B by lower triangular 4 by 4 matrix A, $B \leftarrow AB$, 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

$$B = \begin{pmatrix} -3.0 & -5.0 \\ -1.0 & 1.0 \\ 2.0 & -1.0 \\ 1.0 & 6.0 \end{pmatrix}.$$

9.1 Program Text

```
/* nag_dtrmm (f16yfc) 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 */
  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)*pdb + 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_dtrmm (f16yfc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read the problem dimensions */
  scanf("%ld%ld%*[^\n] ", &m, &n);

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

  /* Read side */
  scanf("%s%*[^\n] ", nag_enum_arg);
  /* 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 */
  scanf("%s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value(x04nac), see above. */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
  /* Read trans */
  scanf("%s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value(x04nac), see above. */
  trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
  /* Read diag */
  scanf("%s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value(x04nac), see above. */
  diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);
  /* Read scalar parameters */
  scanf("%lf%*[^\n] ", &alpha);

  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)
            scanf("%lf", &A(i, j));
        }
      scanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= pda; ++i)
        {
          for (j = 1; j <= i; ++j)
            scanf("%lf", &A(i, j));
        }
      scanf("%*[^\n] ");
    }

  /* Input matrix B */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        scanf("%lf", &B(i, j));
    }

  /* nag_dtrmm(f16yfc).
   * Triangular matrix-matrix multiply.
   *
   */
  nag_dtrmm(order, side, uplo, trans, diag, m, n, alpha, a, pda,
            b, pdb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtrmm (f16yfc).\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:
  if (a) NAG_FREE(a);
  if (b) NAG_FREE(b);

  return exit_status;
}
```

9.2 Program Data

```
nag_dtrmm (f16yfc) 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
 -3.00  -5.00
 -1.00   1.00
  2.00  -1.00
  1.00   6.00                  :End of matrix B
```

9.3 Program Results

```
nag_dtrmm (f16yfc) Example Program Results

 Updated matrix B
             1           2
 1    -12.9000   -21.5000
 2     16.7500    14.9300
 3    -17.5500     6.3300
 4    -11.0400     8.0900
```