NAG C Library Chapter Introduction

f06 – Linear Algebra Support Functions

Contents

1 Scope of the Chapter

This chapter is concerned with basic linear algebra functions which perform elementary algebraic operations involving vectors and matrices.

2 Background to the Problems

All the functions in this chapter meet the specification of the Basic Linear Algebra Subprograms (BLAS) in C as described in Datardina *et al.* (1992). These in turn were derived from the pioneering work of Dongarra *et al.* (1988) and Dongarra *et al.* (1990) on Fortran 77 BLAS. The functions described are concerned with vector operations, matrix-vector operations and matrix-matrix operations. These will be referred to here as the Level-1 BLAS, Level-2 BLAS and Level-3 BLAS respectively. The terminology reflects the number of operations involved. For example, a Level-2 function involves O(n^2) operations for an n by n matrix.

Table 1 indicates the NAG coded naming scheme for the functions in this chapter.

		Level-2	**Level-3**
'real'	BLAS function	f06p_c	f06y_c
'complex'	BLAS function	f06s_c	f06z_c

Table 1

The C BLAS names for these functions are the same as the corresponding Fortran names except that they are in lower case.

The functions in this chapter do not have full function documents, but instead are covered by general descriptions in Section Complex sufficient to enable their use. As this chapter is concerned only with basic linear algebra operations, the functions will not normally be required by the general user. The purpose of each function is indicated in Section 4 so that those users requiring these functions to build specialist linear algebra modules can determine which functions are of interest.

3 Recommendations on Choice and Use of Available Functions

See Section 4 for a list of the functions available for Level-2 (matrix-vector) and Level-3 (matrix-matrix).

3.1 Description of the f06 Functions

The argument lists use the following data types:

`Integer`	an integer data type of at least 32 bits.
`double`	the regular double precision floating-point type.
`Complex`	a double precision complex type.

plus the enumeration types given by

```
typedef enum { NoTranspose, Transpose, ConjugateTranspose } MatrixTranspose;

typedef enum { UpperTriangle, LowerTriangle } MatrixTriangle;

typedef enum { UnitTriangular, NotUnitTriangular } MatrixUnitTriangular;

typedef enum { LeftSide, RightSide } OperationSide;
```

In this section we describe the purpose of each function and give information on the argument lists, where appropriate indicating their general nature. Usually the association between the function arguments and the mathematical variables is obvious and in such cases a description of the argument is omitted.

Within each section, the argument lists for all functions are presented, followed by the purpose of the functions and information on the argument lists.

Within each section functions are listed in alphabetic order of the fifth character in the function name, so that corresponding real and complex functions may have adjacent entries.

3.2 The Level-2 Matrix-vector Functions

The matrix-vector functions all have one array argument representing a matrix; usually this is a two-dimensional array but in some cases the matrix is represented by a one-dimensional array.

The size of the matrix is determined by the arguments **m** and **n** for an m by n rectangular matrix; and by the argument **n** for an n by n symmetric, Hermitian, or triangular matrix. Note that it is permissible to call the functions with **m** or $\mathbf{n} = 0$, in which case the functions exit immediately without referencing their array arguments. For band matrices, the bandwidth is determined by the arguments **kl** and **ku** for a rectangular matrix with **kl** sub-diagonals and **ku** super-diagonals; and by the argument **k** for a symmetric, Hermitian, or triangular matrix with **k** sub-diagonals and/or super-diagonals.

The description of the $m \times n$ matrix consists either of the array name (**a**) followed by the trailing (last) dimension of the array as declared in the calling (sub)program (**tda**), when the matrix is being stored in a two-dimensional array; or the array name (**ap**) alone when the matrix is being stored as a (packed) vector. In the former case the actual array must be allocated at least $((m-1)d+l)$ contiguous elements, where d is the trailing dimension of the array, $d \geq l$, and $l = n$ for arrays representing general, symmetric, Hermitian and triangular matrices, $l = \mathbf{kl} + \mathbf{ku} + 1$ for arrays representing general band matrices and $l = k + 1$ for symmetric, Hermitian and triangular band matrices. For one-dimensional arrays representing matrices (**packed storage**) the actual array must contain at least $\frac{1}{2}n(n+1)$ elements.

The length of each vector, n, is represented by the argument **n**, and the routines may be called with non-positive values of **n**, in which case the routine returns immediately.

In addition to the argument **n**, each vector argument also has an **increment** argument that immediately follows the vector argument, and whose name consists of the three characters **inc**, followed by the name of the vector. For example, a vector x will be represented by the two arguments **x**, **incx**. The increment argument is the spacing (stride) in the array for which the elements of the vector occur. For instance, if $\mathbf{incx} = 2$, then the elements of x are in locations $\mathbf{x}[0], \mathbf{x}[2], \ldots, \mathbf{x}[2 \times \mathbf{n} - 2]$ of the array **x** and the intermediate locations $\mathbf{x}[1], \mathbf{x}[3], \ldots, \mathbf{x}[2 \times \mathbf{n} - 3]$ are not referenced.

Zero increments are not permitted. When $\mathbf{incx} > 0$, the vector element x_i is in the array element $\mathbf{x}[(i-1) \times \mathbf{incx}]$, and when $\mathbf{incx} < 0$ the elements are stored in the reverse order so that the vector element x_i is in the array element $\mathbf{x}[-(n-i) \times \mathbf{incx}]$ and hence, in particular, the element x_n is in $\mathbf{x}[0]$. The declared length of the array **x** in the calling (sub)program must be at least $(1 + (n-1) \times |\mathbf{incx}|)$.

The arguments that specify options are enumeration arguments with the names **trans**, **uplo** and **diag**. **trans** is used by the matrix-vector product functions as follows:

Value	**Meaning**
NoTranspose	Operate with the matrix
Transpose	Operate with the transpose of the matrix
ConjugateTranspose	Operate with the conjugate transpose of the matrix

In the real case the values **Transpose** and **ConjugateTranspose** have the same meaning.

uplo is used by the Hermitian, symmetric, and triangular matrix functions to specify whether the upper or lower triangle is being referenced as follows:

Value	**Meaning**
UpperTriangle	Upper triangle
LowerTriangle	Lower triangle

diag is used by the triangular matrix functions to specify whether or not the matrix is unit triangular, as follows:

Value	**Meaning**
UnitTriangular	Unit triangular
NotUnitTriangular	Non-unit triangular

When **diag** is supplied as **UnitTriangular**, the diagonal elements are not referenced.

3.3 Matrix storage schemes

3.3.1 Conventional storage

The default scheme for storing matrices is the obvious one: a matrix A is stored in a two-dimensional array A, with matrix element a_{ij} stored in array element $A(i,j)$.

If a matrix is **triangular** (upper or lower, as specified by the argument **uplo**), only the elements of the relevant triangle are stored; the remaining elements of the array need not be set. Such elements are indicated by $*$ in the examples below. For example, when $n = 4$:

	Triangular matrix A	**Storage in array a**
uplo = **UpperTriangle**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
uplo = **LowerTriangle**	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

Routines which handle **symmetric** or **Hermitian** matrices allow for either the upper or lower triangle of the matrix (as specified by **uplo**) to be stored in the corresponding elements of the array; the remaining elements of the array need not be set. For example, when $n = 4$:

	Hermitian matrix A	**Storage in array a**
uplo = **UpperTriangle**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
uplo = **LowerTriangle**	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

3.3.2 Packed storage

Symmetric, Hermitian or triangular matrices may be stored more compactly, if the relevant triangle (again as specified by **uplo**) is packed by rows in a one-dimensional array.

if **uplo** = **UpperTriangle**, a_{ij} is stored in **ap**$[j - 1 + (2n - i)(i - 1)/2]$ for $i \leq j$;
if **uplo** = **LowerTriangle**, a_{ij} is stored in **ap**$[j - 1 + i(i - 1)/2]$ for $j \leq i$.

For example:

	Triangle of matrix A	**Packed storage in array ap**
uplo = UpperTriangle	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\underbrace{a_{11}a_{12}a_{13}a_{14}}\underbrace{a_{22}a_{23}a_{24}}\underbrace{a_{33}a_{34}}\underbrace{a_{44}}$
uplo = LowerTriangle	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}}\underbrace{a_{21}a_{22}}\underbrace{a_{31}a_{32}a_{33}}\underbrace{a_{41}a_{42}a_{43}a_{44}}$

Note that for real symmetric matrices, packing the upper triangle by rows is equivalent to packing the lower triangle by columns; packing the lower triangle by rows is equivalent to packing the upper triangle by columns. (For complex Hermitian matrices, the only difference is that the off-diagonal elements are conjugated.)

3.3.3 Band storage

A band matrix with **kl** subdiagonals and **ku** superdiagonals may be stored compactly in a two-dimensional array with **kl** + **ku** + 1 columns and m rows. Rows of the matrix are stored in corresponding rows of the array, and diagonals of the matrix are stored in columns of the array.

For example, when $n = 5$, **kl** = 2 and **ku** = 1:

Band matrix A	**Band storage in array a**
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{11} & a_{12} \\ * & a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} & a_{45} \\ a_{53} & a_{54} & a_{55} & * \end{matrix}$

The elements marked $*$ in the upper left $k_l \times k_l$ triangle and lower right $k_u \times k_u$ of the array **ab** need not be set, and are not referenced by the routines.

The following code fragment will transfer a band matrix $A(m,n)$ from conventional storage to band storage **ab**

```
for(i=0; i<m; ++i){
  k+kl-i;
  for (j=MAX(0,i-kl); j<=MIN(n-1,i+ku); ++j){
    ab[i][k+j]=A[i][j];
  }
}
```

Triangular band matrices are stored in the same format, with either **kl** = 0 if upper triangular, or **ku** = 0 if lower triangular.

For symmetric or Hermitian band matrices with k subdiagonals or superdiagonals, only the upper or lower triangle (as specified by **uplo**) need be stored.

The following code fragments will transfer a symmetric or Hermitian matrix $A(n,n)$ from conventional storage to band storage **ab** if **uplo** = **UpperTriangle**

```
for(i=0; i<n; ++i){
  l=-i;
  for (j=i; j<=MIN(n-1,i+k); ++j){
    ab[i][l+j]=A[i][j];
  }
}
```

if **uplo** = **LowerTriangle**

```
for(i=0; i<n; ++i){
  l=k-i;
  for (j=MAX(0,i-k); j<=i; ++j){
    ab[i][l+j]=A[i][j];
  }
}
```

For example, when $n = 5$ and $k = 2$:

	Hermitian band matrix A	**Band storage in array a**
uplo = **UpperTriangle**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{22} & a_{23} & a_{24} \\ a_{33} & a_{34} & a_{35} \\ a_{44} & a_{45} & * \\ a_{55} & * & * \end{matrix}$
uplo = **LowerTriangle**	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{11} \\ * & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} \\ a_{42} & a_{43} & a_{44} \\ a_{53} & a_{54} & a_{55} \end{matrix}$

Here the elements marked $*$ in the upper left $k \times k$ triangle and the lower right $k \times k$ triangle need not be set and are not referenced by the routines.

3.3.4 Unit triangular matrices

Some routines in this chapter have an option to handle unit triangular matrices (that is, triangular matrices with diagonal elements = 1). This option is specified by an argument **diag**. If **diag** = **UnitTriangular**, the diagonal elements of the matrix need not be stored, and the corresponding array elements are not referenced by the routines. The storage scheme for the rest of the matrix (whether conventional, packed or band) remains unchanged.

3.3.5 Real diagonal elements of complex matrices

Complex Hermitian matrices have diagonal elements that are by definition purely real.

On input only the real parts of the diagonal elements of Hermitian matrices are referenced. The imaginary parts of the diagonals of output Hermitian matrices are set to zero.

3.4 Level-2 BLAS Details of Matrix-vector Operations

Throughout the following sections A^H denotes the complex conjugate of A^T and $\bar{\alpha}$ denotes the complex conjugate of the scalar α.

dgemv (f06pac), zgemv (f06sac), dgbmv (f06pbc) and zgbmv (f06sbc) perform the operation

$$\begin{aligned} y &\leftarrow \alpha Ax + \beta y, && \text{when } \textbf{trans} = \textbf{NoTranspose}, \\ y &\leftarrow \alpha A^T x + \beta y, && \text{when } \textbf{trans} = \textbf{Transpose}, \\ y &\leftarrow \alpha A^H x + \beta y, && \text{when } \textbf{trans} = \textbf{ConjugateTranspose}, \end{aligned}$$

where A is a general matrix for dgemv (f06pac) and zgemv (f06sac), and is a general band matrix for dgbmv (f06pbc) and zgbmv (f06sbc).

dsymv (f06pcc), zhemv (f06scc), dspmv (f06pec), zhpmv (f06sec), dsbmv (f06pdc) and zhbmv (f06sdc) perform the operation

$$y \leftarrow \alpha Ax + \beta y$$

where A is symmetric and Hermitian for dsymv (f06pcc) and zhemv (f06scc) respectively, is symmetric and Hermitian stored in packed form for dspmv (f06pec) and zhpmv (f06sec) respectively, and is symmetric and Hermitian band for dsbmv (f06pdc) and zhbmv (f06sdc).

dtrmv (f06pfc), ztrmv (f06sfc), dtpmv (f06phc), ztpmv (f06shc), dtbmv (f06pgc) and ztbmv (f06sgc) perform the operation

$$\begin{aligned} x &\leftarrow Ax, && \text{when } \textbf{trans} = \textbf{NoTranspose}, \\ x &\leftarrow A^T x, && \text{when } \textbf{trans} = \textbf{Transpose}, \\ x &\leftarrow A^H x, && \text{when } \textbf{trans} = \textbf{ConjugateTranspose}, \end{aligned}$$

where A is a triangular matrix for dtrmv (f06pfc) and ztrmv (f06sfc), is a triangular matrix stored in packed form for dtpmv (f06phc) and ztpmv (f06shc), and is a triangular band matrix for dtbmv (f06pgc) and ztbmv (f06sgc).

dtrsv (f06pjc), ztrsv (f06sjc), dtpsv (f06plc), ztpsv (f06slc), dtbsv (f06pkc) and ztbsv (f06skc) solve the equations

$$\begin{aligned} Ax &= b, && \text{when } \textbf{trans} = \textbf{NoTranspose}, \\ A^T x &= b, && \text{when } \textbf{trans} = \textbf{Transpose}, \\ A^H x &= b, && \text{when } \textbf{trans} = \textbf{ConjugateTranspose}, \end{aligned}$$

where A is a triangular matrix for dtrsv (f06pjc) and ztrsv (f06sjc), is a triangular matrix stored in packed form for dtpsv (f06plc) and ztpsv (f06slc), and is a triangular band matrix for dtbsv (f06pkc) and ztbsv (f06skc). The vector b must be supplied in the array **x** and is overwritten by the solution. It is important to note that no test for singularity is included in these functions.

dger (f06pmc) and zgeru (f06smc) perform the operation $A \leftarrow \alpha x y^T + A$, where A is a general matrix.

zgerc (f06snc) performs the operation $A \leftarrow \alpha x y^H + A$, where A is a general complex matrix.

dsyr (f06ppc) and dspr (f06pqc) perform the operation $A \leftarrow \alpha x x^T + A$, where A is a symmetric matrix for dsyr (f06ppc) and is a symmetric matrix stored in packed form for dspr (f06pqc).

zher (f06spc) and zhpr (f06sqc) perform the operation $A \leftarrow \alpha x x^H + A$, where A is an Hermitian matrix for zher (f06spc) and is an Hermitian matrix stored in packed form for zhpr (f06sqc).

dsyr2 (f06prc) and dspr2 (f06psc) perform the operation $A \leftarrow \alpha x y^T + \alpha y x^T + A$, where A is a symmetric matrix for dsyr2 (f06prc) and is a symmetric matrix stored in packed form for dspr2 (f06psc).

zher2 (f06src) and zhpr2 (f06ssc) perform the operation $A \leftarrow \alpha x y^H + \bar{\alpha} y x^H + A$, where A is an Hermitian matrix for zher2 (f06src) and is an Hermitian matrix stored in packed form for zhpr2 (f06ssc).

The following argument values are invalid:

any value of the enumerated arguments **diag**, **trans**, or **uplo** whose meaning is not specified.

m < 0

n < 0

kl < 0

ku < 0

k < 0

tda $<$ **n** for the functions involving general matrices or full Hermitian, symmetric or triangular matrices **tda** $<$ **kl** $+$ **ku** $+ 1$ for the functions involving general band matrices **tda** $<$ **k** $+ 1$ for the functions involving band Hermitian, symmetric or triangular matrices

incx $= 0$

incy $= 0$

If a function is called with an invalid value then an error message is output on `stderr`, giving the name of the function and the number of the first invalid argument, and execution is terminated.

3.5 The Level-3 Matrix-matrix Functions

The matrix-matrix functions all have either two or three arguments representing a matrix, one of which is an input-output argument, and in each case the arguments are two-dimensional arrays.

The sizes of the matrices are determined by one or more of the arguments **m**, **n** and **k**. The size of the input-output array is always determined by the arguments **m** and **n** for a rectangular m by n matrix, and by the argument **n** for a square n by n matrix. It is permissible to call the functions with **m** or **n** $= 0$, in which case the functions exit immediately without referencing their array arguments.

Many of the functions perform an operation of the form

$$C \leftarrow P + \beta C,$$

where P is the product of two matrices, or the sum of two such products. When the inner dimension of the matrix product is different from m or n it is denoted by **k**. Again it is permissible to call the functions with **k** $= 0$; and if **m** > 0 and **n** > 0, but **k** $= 0$, then the functions perform the operation

$$C \leftarrow \beta C.$$

As with the Level-2 functions (see Section 3.2) the description of the matrix consists of the array name (**a** or **b** or **c**) followed by the second dimension (**tda** or **tdb** or **tdc**).

The arguments that specify options are ennumerated arguments with the names **side**, **transa**, **transb**, **trans**, **uplo** and **diag**. **uplo** and **diag** have the same values and meanings as for the Level-2 functions (see Section 3.2); **transa**, **transb** and **trans** have the same values and meanings as **trans** in the Level-2 functions, where **transa** and **transb** apply to the matrices A and B respectively. **side** is used by the functions as follows:

Value	**Meaning**
LeftSide	Multiply general matrix by symmetric, Hermitian or triangular matrix on the left
Rightside	Multiply general matrix by symmetric, Hermitian or triangular matrix on the right

The storage conventions for matrices are as for the Level-2 functions (see Section 3.2).

3.6 Level-3 BLAS Matrix-matrix Details of Operations

Here, A^H denotes the complex conjugate of A^T and $\bar{\alpha}$ denotes the complex conjugate of the scalar α.

dgemm (f06yac) and zgemm (f06zac) perform the operation indicated in the following table:

	transa $=$ **NoTranspose**	**transa** $=$ **Transpose**	**transa** $=$ **ConjugateTranspose**
transb $=$ **NoTranspose**	$C \leftarrow \alpha AB + \beta C$ A is $m \times k$, B is $k \times n$	$C \leftarrow \alpha A^T B + \beta C$ A is $k \times m$, B is $k \times n$	$C \leftarrow \alpha A^H B + \beta C$ A is $k \times m$ B is $k \times n$
transb $=$ **Transpose**	$C \leftarrow \alpha B^T + \beta C$ A is $m \times k$, B is $n \times k$	$C \leftarrow \alpha A^T B^T + \beta C$ A is $k \times m$, B is $n \times k$	$C \leftarrow \alpha A^H B^T + \beta C$ A is $k \times m$, B is $n \times k$
transb $=$ **ConjugateTranspose**	$C \leftarrow \alpha AB^H + \beta C$ A is $m \times k$, B is $n \times k$	$C \leftarrow \alpha A^T B^H + \beta C$ A is $k \times m$, B is $n \times k$	$C \leftarrow \alpha A^H B^H + \beta C$ A is $k \times m$, B is $n \times k$

where A and B are general matrices and C is a general m by n matrix.

dsymm (f06ycc), zhemm (f06zcc) and zsymm (f06ztc) perform the operation indicated in the following table:

side $=$ **LeftSide**	**side** $=$ **RightSide**
$C \leftarrow \alpha AB + \beta C$	$C \leftarrow \alpha BA + \beta C$
A is $m \times m$	B is $m \times n$
B is $m \times n$	A is $n \times n$

where A is symmetric for dsymm (f06ycc) and zsymm (f06ztc) and is Hermitian for dsymm (f06ycc) and zhemm (f06zcc), B is a general matrix and C is a general m by n matrix.

dtrmm (f06yfc) and ztrmm (f06zfc) perform the operation indicated in the following table:

	transa $=$ **NoTranspose**	**transa** $=$ **Transpose**	**transa** $=$ **ConjugateTranspose**
side $=$ **LeftSide**	$B \leftarrow \alpha AB$ A is triangular $m \times m$	$B \leftarrow \alpha A^T B$ A is triangular $m \times m$	$B \leftarrow \alpha A^H B$ A is triangular $m \times m$

side = **RightSide**	$B \leftarrow \alpha BA$ A is triangular $n \times n$	$B \leftarrow \alpha BA^T$ A is triangular $n \times n$	$B \leftarrow \alpha BA^H$ A is triangular $n \times n$

where B is a general m by n matrix.

dtrsm (f06yjc) and ztrsm (f06zjc) solve the equations, indicated in the following table, for X:

	transa = **NoTranspose**	**transa** = **Transpose**	**transa** = **ConjugateTranspose**
side = **LeftSide**	$AX = \alpha B$ A is triangular $m \times m$	$A^T X = \alpha B$ A is triangular $m \times m$	$A^H X = \alpha B$ A is triangular $m \times m$
side = **RightSide**	$XA = \alpha B$ A is triangular $n \times n$	$XA^T = \alpha B$ A is triangular $n \times n$	$XA^H = \alpha B$ A is triangular $n \times n$

where B is a general m by n matrix. The m by n solution matrix X is overwritten on the array B. It is important to note that no test for singularity is included in these functions.

dsyrk (f06ypc), zherk (f06zpc) and zsyrk (f06zuc) perform the operation indicated in the following table:

	trans = **NoTranspose**	**trans** = **Transpose**	**trans** = **ConjugateTranspose**
dsyrk (f06ypc)	$C \leftarrow \alpha AA^T + \beta C$	$C \leftarrow \alpha A^T A + \beta C$	$C \leftarrow \alpha A^T A + \beta C$
zsyrk (f06zuc)	$C \leftarrow \alpha AA^T + \beta C$	$C \leftarrow \alpha A^T A + \beta C$	–
zherk (f06zpc)	$C \leftarrow \alpha AA^H + \beta C$	–	$C \leftarrow \alpha A^H A + \beta C$
	A is $k \times n$	A is $n \times k$	A is $k \times n$

where A is a general matrix and C is an n by n symmetric matrix for dsyrk (f06ypc) and zsyrk (f06zuc), and is an n by n Hermitian matrix for zherk (f06zpc).

dsyr2k (f06yrc), zher2k (f06zrc) and zsyr2k (f06zwc) perform the operation indicated in the following table:

	trans = **NoTranspose**	**trans** = **Transpose**	**trans** = **ConjugateTranspose**
dsyr2k (f06yrc)	$C \leftarrow \alpha B^T + \alpha BA^T + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
zsyr2k (f06zwc)	$C \leftarrow \alpha B^T + \alpha BA^T + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$	–
zher2k (f06zrc)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C$	–	$C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$
	A and B are $n \times k$	A and B are $k \times n$	A and B are $k \times n$

where A and B are general matrices and C is an n by n symmetric matrix for dsyr2k (f06yrc) and zsyr2k (f06zwc), and is an n by n Hermitian matrix for zherk (f06zpc).

The following values of arguments are invalid:

any value of the ennumerated arguments **side**, **transa**, **transb**, **trans**, **uplo** or **diag**, whose meaning is not specified.

m < 0

n < 0

k < 0

tda $<$ the number of columns in the matrix A.

tdb $<$ the number of columns in the matrix B.

tdc $<$ the number of columnns in the matrix C.

If a function is called with an invalid value, then an error message is output on stderr, giving the name of the function and the number of the first invalid argument, and execution is terminated.

4 Index

Level 1 (Vector) operations:
  Complex vector(s),
    multiply vector by reciprocal of a real scalar .... `nag_zrscl (f06kec)`
  Real vector(s),
    multiply vector by reciprocal of a scalar .... `nag_drscl (f06fec)`

- Level 2 (Matrix-vector and matrix) operations:
  - Complex matrix and vector(s),
    - matrix-vector product,
      - Hermitian band matrix .... `zhbmv (f06sdc)`
      - Hermitian matrix .... `zhemv (f06scc)`
      - Hermitian packed matrix .... `zhpmv (f06sec)`
      - rectangular band matrix .... `zgbmv (f06sbc)`
      - rectangular matrix .... `zgemv (f06sac)`
      - triangular band matrix .... `ztbmv (f06sgc)`
      - triangular matrix .... `ztrmv (f06sfc)`
      - triangular packed matrix .... `ztpmv (f06shc)`
    - rank-1 update,
      - Hermitian matrix .... `zher (f06spc)`
      - Hermitian packed matrix .... `zhpr (f06sqc)`
      - rectangular matrix, conjugated vector .... `zgerc (f06snc)`
      - rectangular matrix, unconjugated vector .... `zgeru (f06smc)`
    - rank-2 update,
      - Hermitian matrix .... `zher2 (f06src)`
      - Hermitian packed matrix .... `zhpr2 (f06ssc)`
    - solution of a system of equations:
      - triangular band matrix .... `ztbsv (f06skc)`
      - triangular matrix .... `ztrsv (f06sjc)`
      - triangular packed matrix .... `ztpsv (f06slc)`
  - Real matrix and vector(s),
    - matrix-vector product,
      - rectangular band matrix .... `dgbmv (f06pbc)`
      - rectangular matrix .... `dgemv (f06pac)`
      - symmetric band matrix .... `dsbmv (f06pdc)`
      - symmetric matrix .... `dsymv (f06pcc)`
      - symmetric packed matrix .... `dspmv (f06pec)`
      - triangular band matrix .... `dtbmv (f06pgc)`
      - triangular matrix .... `dtrmv (f06pfc)`
      - triangular packed matrix .... `dtpmv (f06phc)`
    - rank-1 update,
      - rectangular matrix .... `dger (f06pmc)`
      - symmetric matrix .... `dsyr (f06ppc)`
      - symmetric packed matrix .... `dspr (f06pqc)`
    - rank-2 update,
      - symmetric matrix .... `dsyr2 (f06prc)`
      - symmetric packed matrix .... `dspr2 (f06psc)`
    - solution of system of equations,
      - triangular band matrix .... `dtbsv (f06pkc)`
      - triangular matrix .... `dtrsv (f06pjc)`
      - triangular packed matrix .... `dtpsv (f06plc)`
- Level 3 (Matrix-matrix) operations:
  - Complex matrices:
    - matrix-matrix product:
      - one matrix Hermitian .... `zhemm (f06zcc)`
      - one matrix symmetric .... `zsymm (f06ztc)`
      - triangular matrix .... `ztrmm (f06zfc)`
      - two rectangular matrices .... `zgemm (f06zac)`
    - rank-2k update:
      - of a Hermitian matrix .... `zher2k (f06zrc)`
      - of a symmetric matrix .... `zsyr2k (f06zwc)`
    - rank-k update:
      - of a Hermitian matrix .... `zherk (f06zpc)`
      - of a symmetric matrix .... `zsyrk (f06zuc)`
    - solution of triangular systems of equations .... `ztrsm (f06zjc)`

5 Functions Withdrawn or Scheduled for Withdrawal

None.

6 References

Datardina S P, Du Croz J J, Hammarling S J and Pont M W (1992) A proposed specification of BLAS routines in C *The Journal of C Language Translation* **3** 295–309

Dongarra J J, Du Croz J J, Duff I S and Hammarling S (1990) A set of Level 3 basic linear algebra subprograms *ACM Trans. Math. Software* **16** 1–28

Dongarra J J, Du Croz J J, Hammarling S and Hanson R J (1988) An extended set of FORTRAN basic linear algebra subprograms *ACM Trans. Math. Software* **14** 1–32