Chapter f06 – Linear Algebra Support Functions

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

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 matrix-vector operations and matrix-matrix operations. These will be referred to here as the 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. The Level 1 Blas will be included at a future mark of the C Library.

Table 1.1 indicates the NAG coded naming scheme for the functions in this Chapter.

<table>
<thead>
<tr>
<th>'real' BLAS function</th>
<th>Level-2</th>
<th>'complex' BLAS function</th>
<th>Level-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f06p_c</td>
<td></td>
<td>f06s_c</td>
<td></td>
</tr>
<tr>
<td>f06y_c</td>
<td></td>
<td>f06z_c</td>
<td></td>
</tr>
</tbody>
</table>

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 4 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 3 so that those users requiring these functions to build specialist linear algebra modules can determine which functions are of interest.

3. References


4. Recommendations on Choice and Use of Functions

This section lists the functions in the categories Level-2 (matrix-vector) and Level-3 (matrix-matrix). The corresponding BLAS name is indicated in brackets.

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

4.1. The Level-2 Matrix-vector Functions

The Level-2 functions perform matrix-vector operations, such as forming the product between a matrix and a vector.
Compute a matrix-vector product; real general matrix \( \text{dgemv} (\text{f06pac}) \)
Compute a matrix-vector product; complex general matrix \( \text{zgemv} (\text{f06sac}) \)
Compute a matrix-vector product; real general band matrix \( \text{dgbmv} (\text{f06pbc}) \)
Compute a matrix-vector product; complex general band matrix \( \text{zgbmv} (\text{f06abc}) \)
Compute a matrix-vector product; real symmetric matrix \( \text{dsymv} (\text{f06pcc}) \)
Compute a matrix-vector product; complex Hermitian matrix \( \text{zhemv} (\text{f06scc}) \)
Compute a matrix-vector product; real symmetric band matrix \( \text{dsbmv} (\text{f06adc}) \)
Compute a matrix-vector product; complex Hermitian band matrix \( \text{zhbmv} (\text{f06adc}) \)
Compute a matrix-vector product; real symmetric packed matrix \( \text{dspmv} (\text{f06pec}) \)
Compute a matrix-vector product; complex Hermitian packed matrix \( \text{zhpmv} (\text{f06sec}) \)
Compute a matrix-vector product; real triangular matrix \( \text{dtrmv} (\text{f06pfc}) \)
Compute a matrix-vector product; complex triangular matrix \( \text{ztrmv} (\text{f06afc}) \)
Compute a matrix-vector product; real triangular band matrix \( \text{dtbmv} (\text{f06pgc}) \)
Compute a matrix-vector product; complex triangular band matrix \( \text{ztbmv} (\text{f06agc}) \)
Compute a matrix-vector product; real triangular packed matrix \( \text{dtpmv} (\text{f06phc}) \)
Compute a matrix-vector product; complex triangular packed matrix \( \text{ztzmv} (\text{f06ahc}) \)
Solve a system of equations; real triangular coefficient matrix \( \text{dtrsv} (\text{f06pjc}) \)
Solve a system of equations; complex triangular coefficient matrix \( \text{ztrsv} (\text{f06ajc}) \)
Solve a system of equations; real triangular band coefficient matrix \( \text{dtbsv} (\text{f06pkc}) \)
Solve a system of equations; complex triangular band coefficient matrix \( \text{ztbsv} (\text{f06akc}) \)
Solve a system of equations; real triangular packed coefficient matrix \( \text{dtpsv} (\text{f06plc}) \)
Solve a system of equations; complex triangular packed coefficient matrix \( \text{ztzsv} (\text{f06alc}) \)
Perform a rank-one update; real general matrix \( \text{dger} (\text{f06pmc}) \)
Perform a rank-one update; complex general matrix (unconjugated vector) \( \text{zgeru} (\text{f06pmc}) \)
Perform a rank-one update; complex general matrix (conjugated vector) \( \text{zgerc} (\text{f06pmc}) \)
Perform a rank-one update; real symmetric matrix \( \text{dsyr} (\text{f06ppc}) \)
Perform a rank-one update; complex Hermitian matrix \( \text{zher} (\text{f06ppc}) \)
Perform a rank-one update; real symmetric packed matrix \( \text{dspr} (\text{f06ppc}) \)
Perform a rank-one update; complex Hermitian packed matrix \( \text{zhpr} (\text{f06ppc}) \)
Perform a rank-two update; real symmetric matrix \( \text{dsyr2} (\text{f06ppc}) \)
Perform a rank-two update; complex Hermitian matrix \( \text{zher2} (\text{f06ppc}) \)
Perform a rank-two update; real symmetric packed matrix \( \text{dspr2} (\text{f06ppc}) \)
Perform a rank-two update; complex Hermitian packed matrix \( \text{zhpr2} (\text{f06ppc}) \)

**4.2. The Level-3 Matrix-matrix Functions**

The Level-3 functions perform matrix-matrix operations, such as forming the product of two matrices.

Compute a matrix-matrix product; two real rectangular matrices \( \text{dgemm} (\text{f06yac}) \)
Compute a matrix-matrix product; two complex rectangular matrices \( \text{zgemm} (\text{f06zac}) \)
Compute a matrix-matrix product; one real symmetric matrix, one real rectangular matrix \( \text{dsymm} (\text{f06ycc}) \)
Compute a matrix-matrix product; one complex Hermitian matrix, one complex rectangular matrix \( \text{zhemm} (f06zcc) \)

Compute a matrix-matrix product; one real triangular matrix, one real rectangular matrix \( \text{dtrmm} (f06yfc) \)

Compute a matrix-matrix product; one complex triangular matrix, one complex rectangular matrix \( \text{ztrmm} (f06zfc) \)

Solve a system of equations with multiple right-hand sides, real triangular coefficient matrix \( \text{dtrsm} (f06yjc) \)

Solve a system of equations with multiple right-hand sides, complex triangular coefficient matrix \( \text{ztrsm} (f06zjc) \)

Perform a rank-\( k \) update of a real symmetric matrix \( \text{dsyrk} (f06ypc) \)

Perform a rank-\( k \) update of a complex Hermitian matrix \( \text{zherk} (f06zpc) \)

Perform a rank-2\( k \) update of a real symmetric matrix \( \text{dsyr2k} (f06yrc) \)

Perform a rank-2\( k \) update of a complex Hermitian matrix \( \text{zher2k} (f06zrc) \)

Compute a matrix-matrix product; one complex symmetric matrix, one complex rectangular matrix \( \text{zsymm} (f06ztc) \)

Perform a rank-\( k \) update of a complex symmetric matrix \( \text{zsyrk} (f06zuc) \)

Perform a rank-2\( k \) update of a complex symmetric matrix \( \text{zsyr2k} (f06zwc) \)

5. 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

```c
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.

5.1. 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 \( m = 0 \) or \( n = 0 \), 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, \( l = kl + ku + 1 \) 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 \( incx = 2 \), then the elements of \( x \) are in locations \( x[0], x[2], \ldots, x[2 \ast n - 2] \) of the array \( x \) and the intermediate locations \( x[1], x[3], \ldots, x[2 \ast n - 3] \) are not referenced.

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

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoTranspose</td>
<td>Operate with the matrix</td>
</tr>
<tr>
<td>Transpose</td>
<td>Operate with the transpose of the matrix</td>
</tr>
<tr>
<td>ConjugateTranspose</td>
<td>Operate with the conjugate transpose of the matrix</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpperTriangle</td>
<td>Upper triangle</td>
</tr>
<tr>
<td>LowerTriangle</td>
<td>Lower triangle</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnitTriangular</td>
<td>Unit triangular</td>
</tr>
<tr>
<td>NotUnitTriangular</td>
<td>Non-unit triangular</td>
</tr>
</tbody>
</table>

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

5.1.1. Matrix storage schemes

Conventional storage

The default scheme for storing matrices is the obvious one: a matrix \( A \) is stored in a 2-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 \( n = 4 \):

<table>
<thead>
<tr>
<th><code>uplo</code></th>
<th>Triangular matrix A</th>
<th>Storage in array A</th>
</tr>
</thead>
</table>
| **UpperTriangle** | \[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{22} & a_{23} & a_{24} & \\
  a_{33} & a_{34} & \\
  a_{44} & 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  * & a_{22} & a_{23} & a_{24} \\
  * & * & a_{33} & a_{34} \\
  * & * & * & a_{44} 
\end{pmatrix}
\] |
| **LowerTriangle** | \[
\begin{pmatrix}
  a_{11} & a_{21} & a_{31} & a_{41} \\
  a_{22} & a_{23} & a_{32} & a_{42} \\
  a_{33} & a_{34} & a_{43} \\
  a_{44} & 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & * & * & * \\
  a_{21} & a_{22} & * & * \\
  a_{31} & a_{32} & a_{33} & * \\
  a_{41} & a_{42} & a_{43} & a_{44} 
\end{pmatrix}
\] |

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 \( n = 4 \):

<table>
<thead>
<tr>
<th><code>uplo</code></th>
<th>Hermitian matrix A</th>
<th>Storage in array A</th>
</tr>
</thead>
</table>
| **UpperTriangle** | \[
\begin{pmatrix}
  a_{11} & \bar{a}_{12} & a_{13} & a_{14} \\
  \bar{a}_{12} & \bar{a}_{22} & \bar{a}_{23} & \bar{a}_{24} \\
  \bar{a}_{13} & \bar{a}_{23} & \bar{a}_{33} & \bar{a}_{34} \\
  \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & \bar{a}_{44} 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  * & a_{22} & a_{23} & a_{24} \\
  * & * & a_{33} & a_{34} \\
  * & * & * & a_{44} 
\end{pmatrix}
\] |
| **LowerTriangle** | \[
\begin{pmatrix}
  a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\
  \bar{a}_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\
  \bar{a}_{31} & \bar{a}_{32} & a_{33} & \bar{a}_{43} \\
  \bar{a}_{41} & \bar{a}_{42} & \bar{a}_{43} & a_{44} 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & * & * & * \\
  a_{21} & a_{22} & * & * \\
  a_{31} & a_{32} & a_{33} & * \\
  a_{41} & a_{42} & a_{43} & a_{44} 
\end{pmatrix}
\] |

**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 1-dimensional array.

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

For example:

<table>
<thead>
<tr>
<th><code>uplo</code></th>
<th>Triangular matrix a</th>
<th>Packed storage in array ap</th>
</tr>
</thead>
</table>
| **UpperTriangle** | \[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{22} & a_{23} & a_{24} & \\
  a_{33} & a_{34} & \\
  a_{44} & 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{22} & a_{23} & a_{24} & \\
  a_{33} & a_{34} & \\
  a_{44} & 
\end{pmatrix}
\] |
| **LowerTriangle** | \[
\begin{pmatrix}
  a_{11} & a_{21} & a_{31} & a_{41} \\
  a_{21} & a_{22} & a_{32} & a_{42} \\
  a_{31} & a_{32} & a_{33} & \\
  a_{41} & a_{42} & a_{43} & a_{44} 
\end{pmatrix}
\] | \[
\begin{pmatrix}
  a_{11} & a_{21} & a_{31} & a_{41} \\
  a_{22} & a_{32} & a_{33} & \\
  a_{42} & a_{43} & a_{44} 
\end{pmatrix}
\] |
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.)

Band storage

A band matrix with $kl$ subdiagonals and $ku$ superdiagonals may be stored compactly in a 2-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 $ku = 1$:

<table>
<thead>
<tr>
<th>Band Matrix $a$</th>
<th>Band storage in array $ab$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{11}$ $a_{12}$  $a_{21}$ $a_{22}$ $a_{23}$  $a_{31}$ $a_{32}$ $a_{33}$ $a_{34}$ $a_{35}$  $a_{42}$ $a_{43}$ $a_{44}$ $a_{45}$  $a_{53}$ $a_{54}$ $a_{55}$</td>
<td>$<em>$  $</em>$ $a_{11}$ $a_{12}$  $a_{21}$ $a_{22}$ $a_{23}$  $a_{31}$ $a_{32}$ $a_{33}$ $a_{34}$ $a_{35}$  $a_{42}$ $a_{43}$ $a_{44}$ $a_{45}$  $a_{53}$ $a_{54}$ $a_{55}$ $*$</td>
</tr>
</tbody>
</table>

The elements marked $*$ in the upper left $kl \times kl$ triangle and lower right $ku \times ku$ 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 $ab$

```c
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 $ab$

```c
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 $k = 2$: 
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.

**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 \texttt{diag}. If \texttt{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.

**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.

### 5.1.2. Level-2 BLAS Functions Specification

In the following specifications, the argument \texttt{ap} refers to arrays containing matrices in packed storage order.

- \texttt{void dgemv} (\texttt{MatrixTranspose trans}, \texttt{Integer m}, \texttt{Integer n}, \texttt{double alpha}, \texttt{const double a[]}, \texttt{Integer tda}, \texttt{const double x[]}, \texttt{Integer incx}, \texttt{double beta}, \texttt{double y[]}, \texttt{Integer incy})

- \texttt{void zgemv} (\texttt{MatrixTranspose trans}, \texttt{Integer m}, \texttt{Integer n}, \texttt{Complex alpha}, \texttt{const Complex a[]}, \texttt{Integer tda}, \texttt{const Complex x[]}, \texttt{Integer incx}, \texttt{Complex beta}, \texttt{Complex y[]}, \texttt{Integer incy})

- \texttt{void dgbmv}(\texttt{MatrixTranspose trans}, \texttt{Integer m}, \texttt{Integer n}, \texttt{double alpha}, \texttt{const double a[]}, \texttt{Integer tda}, \texttt{const double x[]}, \texttt{Integer incx}, \texttt{double beta}, \texttt{double y[]}, \texttt{Integer incy})

- \texttt{void zgbmv}(\texttt{MatrixTranspose trans}, \texttt{Integer m}, \texttt{Integer n}, \texttt{Complex alpha}, \texttt{const Complex a[]}, \texttt{Integer tda}, \texttt{const Complex x[]}, \texttt{Integer incx}, \texttt{Complex beta}, \texttt{Complex y[]}, \texttt{Integer incy})

- \texttt{void dsymv}(\texttt{MatrixTriangle uplo}, \texttt{Integer n}, \texttt{double alpha}, \texttt{const double a[]}, \texttt{Integer tda}, \texttt{const double x[]}, \texttt{Integer incx}, \texttt{double beta}, \texttt{double y[]}, \texttt{Integer incy})

- \texttt{void zhemv}(\texttt{MatrixTriangle uplo}, \texttt{Integer n}, \texttt{Complex alpha}, \texttt{const Complex a[]}, \texttt{Integer tda}, \texttt{const Complex x[]}, \texttt{Integer incx}, \texttt{Complex beta}, \texttt{Complex y[]}, \texttt{Integer incy})
void dsbmv(MatrixTriangle uplo, Integer n, Integer k,
double alpha, const double a[], Integer tda,
const double x[], Integer incx, double beta,
double y[], Integer incy)  \[\text{f06pdc}\]

void zhbmv(MatrixTriangle uplo, Integer n, Integer k,
Complex alpha, const Complex a[], Integer tda,
const Complex x[], Integer incx, Complex beta,
Complex y[], Integer incy)  \[\text{f06sdc}\]

void dspmv(MatrixTriangle uplo, Integer n, double alpha,
const double ap[], const double x[], Integer incx,
double beta, double y[], Integer incy)  \[\text{f06pec}\]

void zhpmv(MatrixTriangle uplo, Integer n, Complex alpha,
const Complex ap[], const Complex x[],
Integer incx, Complex beta, Complex y[],
Integer incy)  \[\text{f06sec}\]

void dtrmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const double a[], Integer tda, double x[])
\[\text{f06pfc}\]

void ztrmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const Complex a[], Integer tda, Complex x[])
\[\text{f06sfc}\]

void dtbmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n, Integer k,
const double a[], Integer tda, double x[])
\[\text{f06pgc}\]

void ztbmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n, Integer k,
const Complex a[], Integer tda, Complex x[])
\[\text{f06sgc}\]

void dtpmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const double ap[], double x[], Integer incx)
\[\text{f06phc}\]

void ztpmv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const Complex ap[], Complex x[], Integer incx)
\[\text{f06shc}\]

void dtrsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const double a[], Integer tda, double x[])
\[\text{f06pjc}\]

void ztrsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const Complex a[], Integer tda, Complex x[])
\[\text{f06sjc}\]

void dtbsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n, Integer k,
const double a[], Integer tda, double x[])
\[\text{f06pkc}\]

void ztbsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n, Integer k,
const Complex a[], Integer tda, Complex x[])
\[\text{f06skc}\]

void dtpsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const double ap[], double x[], Integer incx)
\[\text{f06plc}\]

void ztpsv(MatrixTriangle uplo, MatrixTranspose trans,
MatrixUnitTriangular diag, Integer n,
const Complex ap[], Complex x[], Integer incx)
\[\text{f06slc}\]

void dger(Integer m, Integer n, double alpha,
const double x[], Integer incx, const double y[],
Integer incy, double a[], Integer tda)  \[\text{f06pmc}\]
5.1.3. Level-2 BLAS Details of Matrix-vector Operations

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

$f06pac$, $f06sac$, $f06pbc$ and $f06sbc$
perform the operation

\[
y \leftarrow \alpha Ax + \beta y, \quad \text{when } trans=\text{NoTranspose},
\]
\[
y \leftarrow \alpha A^T x + \beta y, \quad \text{when } trans=\text{Transpose},
\]
\[
y \leftarrow \alpha A^H x + \beta y, \quad \text{when } trans=\text{ConjugateTranspose},
\]

where $A$ is a general matrix for $f06pac$ and $f06sac$, and is a general band matrix for $f06pbc$ and $f06sbc$.

$f06pcc$, $f06scc$, $f06pec$, $f06sec$, $f06pdc$ and $f06sdc$
perform the operation

\[
y \leftarrow \alpha Ax + \beta y
\]

where $A$ is symmetric and Hermitian for $f06pcc$ and $f06scc$ respectively, is symmetric and Hermitian stored in packed form for $f06pec$ and $f06sec$ respectively, and is symmetric and Hermitian band for $f06pdc$ and $f06sdc$.

$f06pfc$, $f06sfc$, $f06phc$, $f06shc$, $f06pgc$ and $f06sgc$
perform the operation

\[
y \leftarrow \alpha Ax + \beta y
\]
$x \leftarrow Ax$, when $\text{trans}=\text{Notranspose}$,
$x \leftarrow A^T x$, when $\text{trans}=\text{Transpose}$,
$x \leftarrow A^H x$, when $\text{trans}=\text{ConjugateTranspose}$,

where $A$ is a triangular matrix for $f06pfc$ and $f06sc$, is a triangular matrix stored in packed form for $f06pfc$ and $f06sc$, and is a triangular band matrix for $f06pgc$ and $f06sgc$.

$f06pjc$, $f06sjc$, $f06plc$, $f06slc$, $f06pkc$ and $f06skc$
solve the equations

$Ax = b$, when $\text{trans}=\text{Notranspose}$,
$A^T x = b$, when $\text{trans}=\text{Transpose}$,
$A^H x = b$, when $\text{trans}=\text{ConjugateTranspose}$,

where $A$ is a triangular matrix for $f06pjc$ and $f06sjc$, is a triangular matrix stored in packed form for $f06plc$ and $f06slc$, and is a triangular band matrix for $f06pkc$ and $f06skc$. 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.

$f06pmc$ and $f06smc$
perform the operation $A \leftarrow \alpha xy^T + A$, where $A$ is a general matrix.

$f06snc$
performs the operation $A \leftarrow \alpha xy^H + A$, where $A$ is a general complex matrix.

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

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

$f06prc$ and $f06psc$
perform the operation $A \leftarrow \alpha xy^T + \alpha yx^T + A$, where $A$ is a symmetric matrix for $f06prc$ and is a symmetric matrix stored in packed form for $f06psc$.

$f06src$ and $f06ssc$
perform the operation $A \leftarrow \alpha xy^H + \alpha yx^H + A$, where $A$ is an Hermitian matrix for $f06src$ and is an Hermitian matrix stored in packed form for $f06ssc$.

The following argument values are invalid:

Any value of the enumerated arguments $\text{diag}$, $\text{trans}$, or $\text{uplo}$ 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
- $\text{incx} = 0$
- $\text{incy} = 0$
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.

5.2. 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 \( m \times n \) matrix, and by the argument \( n \) for a square \( n \times n \) matrix. It is permissible to call the functions with \( m = 0 \) 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 4.1) the description of the matrix consists of the array name (\( \text{a} \) or \( \text{b} \) or \( \text{c} \)) followed by the second dimension (\( \text{tda} \) or \( \text{tdb} \) or \( \text{tdc} \)).

The arguments that specify options are enumerated arguments with the names \( \text{side} \), \( \text{transa} \), \( \text{transb} \), \( \text{trans} \), \( \text{uplo} \) and \( \text{diag} \). \( \text{uplo} \) and \( \text{diag} \) have the same values and meanings as for the Level-2 functions (see Section 4.1); \( \text{transa} \) and \( \text{transb} \) have the same values and meanings as \( \text{trans} \) in the Level-2 functions, where \( \text{transa} \) and \( \text{transb} \) apply to the matrices \( A \) and \( B \) respectively. \( \text{side} \) is used by the functions as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{LeftSide}</td>
<td>Multiply general matrix by symmetric, Hermitian or triangular matrix on the left</td>
</tr>
<tr>
<td>\text{Rightside}</td>
<td>Multiply general matrix by symmetric, Hermitian or triangular matrix on the right</td>
</tr>
</tbody>
</table>

The storage conventions for matrices are as for the Level-2 functions (see Section 4.1).

5.2.1. Level-3 BLAS Functions Specification

```c
void dgemm(MatrixTranspose transa, MatrixTranspose transb, 
            Integer m, Integer n, Integer k, double alpha, 
            const double a[], Integer tda, const double b[], 
            Integer tdb, double beta, double c[], Integer tdc)

void zgemm(MatrixTranspose transa, MatrixTranspose transb, 
            Integer m, Integer n, Integer k, Complex alpha, 
            const Complex a[], Integer tda, const Complex b[], 
            Integer tdb, Complex beta, Complex c[], 
            Integer tdc)

void dsymm(OperationSide side, MatrixTriangle uplo, 
            Integer m, Integer n, double alpha, 
            const double a[], Integer tda, const double b[], 
            Integer tdb, double beta, double c[], Integer tdc)

void zhemm(OperationSide side, MatrixTriangle uplo, 
            Integer m, Integer n, Complex alpha, 
            const Complex a[], Integer tda, const Complex b[], 
            Integer tdb, Complex beta, Complex c[], 
            Integer tdc)
```

[NP3275/5/pdf] 3.intro-f06.11
void dtrmm(MatrixTriangle side, MatrixTriangle uplo,
MatrixTranspose transa, MatrixUnitTriangular diag,
Integer m, Integer n, double alpha,
const double a[], Integer tda, double b[],
Integer tdb)

void ztrmm(MatrixTriangle side, MatrixTriangle uplo,
MatrixTranspose transa, MatrixUnitTriangular diag,
Integer m, Integer n, Complex alpha,
const Complex a[], Integer tda, Complex b[],
Integer tdb)

void dtrsm(OperationSide side, MatrixTriangle uplo,
MatrixTranspose transa, MatrixUnitTriangular diag,
Integer m, Integer n, double alpha,
const double a[], Integer tda, double b[],
Integer tdb)

void ztrsm(OperationSide side, MatrixTriangle uplo,
MatrixTranspose transa, MatrixUnitTriangular diag,
Integer m, Integer n, Complex alpha,
const Complex a[], Integer tda, Complex b[],
Integer tdb)

void dsyrk(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, double alpha,
const double a[], Integer tda, double beta,
double c[], Integer tdc)

void zherk(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, double alpha,
const Complex a[], Integer tda, Complex c[],
Integer tdc)

void dsyr2k(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, double alpha,
const double a[], Integer tda, const double b[],
Integer tdb, double beta, double c[], Integer tdc)

void zher2k(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, Complex alpha,
const Complex a[], Integer tda,
const Complex b[], Integer tdb, Complex beta,
Complex c[], Integer tdc)

void zsymm(OperationSide side, MatrixTriangle uplo,
Integer m, Integer n, Complex alpha,
const Complex a[], Integer tda, const Complex b[],
Complex beta, Complex c[],
Integer tdc)

void zsyrk(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, Complex alpha,
const Complex a[], Integer tda, Complex beta,
Complex c[], Integer tdc)

void zsyr2k(MatrixTriangle uplo, MatrixTranspose trans,
Integer n, Integer k, Complex alpha,
const Complex a[], Integer tda, const Complex b[],
Integer tdb, Complex beta, Complex c[],
Integer tdc)

5.2.2. Level-3 BLAS Matrix-matrix Details of Operations

Here as in Section 4.1.2, $A^H$ denotes the complex conjugate of $A^T$ and $\bar{\alpha}$ denotes the complex conjugate of the scalar $\alpha$.

f06yac and f06zac

perform the operation indicated in the following table:
where $A$ and $B$ are general matrices and $C$ is a general $m \times n$ matrix.

$f06yc$, $f06zcc$ and $f06ztc$ perform the operation indicated in the following table:

<table>
<thead>
<tr>
<th>transa=Notranspose</th>
<th>transa=Transpose</th>
<th>transa=ConjugateTranspose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{transb=Notranspose}$</td>
<td>$C \leftarrow \alpha AB + \beta C$</td>
<td>$C \leftarrow \alpha A^T B + \beta C$</td>
</tr>
<tr>
<td>$\text{transb=Transpose}$</td>
<td>$C \leftarrow \alpha A^T B + \beta C$</td>
<td>$C \leftarrow \alpha A^T B^T + \beta C$</td>
</tr>
<tr>
<td>$\text{transb=ConjugateTranspose}$</td>
<td>$C \leftarrow \alpha A^H B + \beta C$</td>
<td>$C \leftarrow \alpha A^T B^H + \beta C$</td>
</tr>
</tbody>
</table>

where $A$ is symmetric for $f06yc$ and $f06ztc$ and is Hermitian for $f06zcc$, $B$ is a general matrix and $C$ is a general $m \times n$ matrix.

$f06yfc$ and $f06zfc$ perform the operation indicated in the following table:

<table>
<thead>
<tr>
<th>transa=Notranspose</th>
<th>transa=Transpose</th>
<th>transa=ConjugateTranspose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{side=Leftside}$</td>
<td>$B \leftarrow \alpha AB$</td>
<td>$B \leftarrow \alpha A^T B$</td>
</tr>
<tr>
<td>$\text{side=Rightside}$</td>
<td>$B \leftarrow \alpha AB$</td>
<td>$B \leftarrow \alpha A^T B$</td>
</tr>
</tbody>
</table>

where $B$ is a general $m \times n$ matrix.

$f06yjc$ and $f06zjc$ solve the equations, indicated in the following table, for $X$:

<table>
<thead>
<tr>
<th>transa=Notranspose</th>
<th>transa=Transpose</th>
<th>transa=ConjugateTranspose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{side=Leftside}$</td>
<td>$AX = \alpha B$</td>
<td>$A^T X = \alpha B$</td>
</tr>
<tr>
<td>$\text{side=Rightside}$</td>
<td>$XA = \alpha B$</td>
<td>$XA^T = \alpha B$</td>
</tr>
</tbody>
</table>

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

$f06ypc$, $f06zpc$ and $f06zuc$ perform the operation indicated in the following table:

<table>
<thead>
<tr>
<th>transa=Notranspose</th>
<th>transa=Transpose</th>
<th>transa=ConjugateTranspose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{f06ypc}$</td>
<td>$C \leftarrow \alpha AA^T + \beta C$</td>
<td>$C \leftarrow \alpha A^T A + \beta C$</td>
</tr>
<tr>
<td>$\text{f06zuc}$</td>
<td>$C \leftarrow \alpha AA^T + \beta C$</td>
<td>$C \leftarrow \alpha A^T A + \beta C$</td>
</tr>
<tr>
<td>$\text{f06zpc}$</td>
<td>$C \leftarrow \alpha AA^H + \beta C$</td>
<td>$C \leftarrow \alpha A^T A + \beta C$</td>
</tr>
</tbody>
</table>

where $A$ is $n \times k$, $A$ is $k \times n$, $A$ is $k \times n$. 

13
where A is a general matrix and C is an n by n symmetric matrix for f06ypc and f06zuc, and is an n by n Hermitian matrix for f06zpc.

f06yrc, f06zrc and f06zwc perform the operation indicated in the following table:

<table>
<thead>
<tr>
<th>trans=Notranspose</th>
<th>trans=Transpose</th>
<th>trans=ConjugateTranspose</th>
</tr>
</thead>
<tbody>
<tr>
<td>f06yrc</td>
<td>C ← αABᵀ + αBAᵀ + βC</td>
<td>C ← αAᵀB + αBᵀA + βC</td>
</tr>
<tr>
<td>f06zwc</td>
<td>C ← αABᵀ + αBAᵀ + βC</td>
<td>C ← αAᵀB + αBᵀA + βC</td>
</tr>
<tr>
<td>f06zrc</td>
<td>C ← αABᴴ + αBAᴴ + βC</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>A and B are n × k</td>
<td>A ← αAᴴB + αBᴴA + βC</td>
</tr>
</tbody>
</table>

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

The following values of arguments are invalid:

Any value of the enumerated 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 columns 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.