

NAG Library Chapter Introduction

F06 – Linear Algebra Support Routines

Contents

1	Scope of the Chapter	2
2	Background to the Problems	2
2.1	The Use of BLAS Names	2
2.2	Background Information	2
2.2.1	Real plane rotations	2
2.2.2	Complex plane rotations	4
2.2.3	Elementary real (Householder) reflections	4
2.2.4	Elementary complex (Householder) reflections	5
3	Recommendations on Choice and Use of Available Routines	6
3.1	Naming Scheme	6
3.1.1	NAG names	6
3.1.2	BLAS names	6
3.2	The Level-0 Scalar Routines	7
3.3	The Level-1 Vector Routines	7
3.4	The Level-2 Matrix-vector and Matrix Routines	7
3.5	The Level-3 Matrix-matrix Routines	7
3.6	Vector Arguments	7
3.7	Matrix Arguments and Storage Schemes	8
3.7.1	Conventional storage	8
3.7.2	Packed storage	9
3.7.3	Band storage	9
3.7.4	Unit triangular matrices	10
3.7.5	Real diagonal elements of complex Hermitian matrices	10
3.7.6	Spiked matrices	10
3.8	Option Parameters	11
3.8.1	Matrix norms	12
3.9	Error Handling	12
4	Index	12
5	Routines Withdrawn or Scheduled for Withdrawal	17
6	References	17

1 Scope of the Chapter

This chapter is concerned with basic linear algebra routines which perform elementary algebraic operations involving scalars, vectors and matrices. It includes routines which conform to the specifications of the BLAS (Basic Linear Algebra Subprograms).

2 Background to the Problems

A number of the routines in this chapter meet the specification of the Basic Linear Algebra Subprograms (BLAS) as described in Lawson *et al.* (1979), Dodson *et al.* (1991), Dongarra *et al.* (1988) and Dongarra *et al.* (1990). The first reference describes a set of routines concerned with operations on scalars and vectors: these will be referred to here as the Level-0 and the Level-1 BLAS; the second reference describes a set of routines concerned with operations on sparse vectors: these will be referred to here as the Level-1 Sparse BLAS; the third reference describes a set of routines concerned with matrix-vector operations: these will be referred to here as the Level-2 BLAS; and the fourth reference describes a set of routines concerned with matrix-matrix operations: these will be referred to here as the Level-3 BLAS.

More generally we refer to the scalar routines in the chapter as Level-0 routines, to the vector routines as Level-1 routines, to the matrix-vector and matrix routines as Level-2 routines, and to the matrix-matrix routines as Level-3 routines. The terminology reflects the number of operations involved. For example, a Level-2 routine involves $O(n^2)$ operations for an $n \times n$ matrix.

2.1 The Use of BLAS Names

Many of the routines in other chapters of the Library call the routines in this chapter, and in particular a number of the BLAS are called. These routines are usually called by the BLAS name and so, for correct operation of the Library, it is essential that you do not attempt to link your own versions of these routines. If you are in any doubt about how to avoid this, please consult your computer centre or the NAG Response Centre.

The BLAS names are used in order to make use of efficient implementations of the routines when these exist. Such implementations are stringently tested before being used, to ensure that they correctly meet the specification of the BLAS, and that they return the desired accuracy (see, for example, Dodson *et al.* (1991), Dongarra *et al.* (1988) and Dongarra *et al.* (1990)).

2.2 Background Information

Most of the routines in this chapter implement straightforward scalar, vector and matrix operations that need no further explanation beyond a statement of the purpose of the routine. In this section we give some additional background information to those few cases where additional explanation may be necessary. A sub-section is devoted to each topic.

2.2.1 Real plane rotations

There are a number of routines in the chapter concerned with setting up and applying plane rotations. This section discusses the real case and the next section looks at the complex case. For further background information see Golub and Van Loan (1996).

A plane rotation matrix for the (i, j) plane, R_{ij} , is an orthogonal matrix that is different from the unit matrix only in the elements r_{ii} , r_{jj} , r_{ij} and r_{ji} . If we put

$$R = \begin{pmatrix} r_{ii} & r_{ij} \\ r_{ji} & r_{jj} \end{pmatrix}, \quad (1)$$

then, in the real case, it is usual to choose R_{ij} so that

$$R = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \quad c = \cos \theta, \quad s = \sin \theta.$$

An exception is routine F06FPF which applies the so-called symmetric rotation for which

$$R = \begin{pmatrix} c & s \\ s & -c \end{pmatrix}. \quad (2)$$

The application of plane rotations is straightforward and needs no further elaboration, so further comment is made only on the construction of plane rotations.

The most common use of plane rotations is to choose c and s so that for given a and b ,

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}. \quad (3)$$

In such an application the matrix R is often termed a **Givens rotation** matrix. There are two approaches to the construction of real Givens rotations in Chapter F06.

The BLAS routine F06AAF (DROTG), see Lawson *et al.* (1979) and Dodson and Grimes (1982), computes c , s and d as

$$d = \sigma(a^2 + b^2)^{1/2},$$

$$c = \begin{cases} a/d, & d \neq 0, \\ 1, & d = 0, \end{cases} \quad s = \begin{cases} b/d, & d \neq 0, \\ 0, & d = 0, \end{cases} \quad (4)$$

where $\sigma = \begin{cases} \text{sign } a, & |a| > |b| \\ \text{sign } b, & |a| \leq |b|. \end{cases}$

The value z defined as

$$z = \begin{cases} s, & |s| < c \quad \text{or} \quad c = 0 \\ 1/c, & 0 < |c| \leq s \end{cases} \quad (5)$$

is also computed and this enables c and s to be reconstructed from the single value z as

$$c = \begin{cases} 0, & z = 1 \\ (1 - z^2)^{1/2}, & |z| < 1 \\ 1/z, & |z| > 1 \end{cases} \quad s = \begin{cases} 1, & z = 1 \\ z, & |z| < 1 \\ (1 - c^2)^{1/2}, & |z| > 1 \end{cases}.$$

The other Chapter F06 routines for constructing Givens rotations are based on the computation of the tangent, $t = \tan \theta$. t is computed as

$$t = \begin{cases} 0, & b = 0 \\ b/a, & |b| \leq |a|.flmax, b \neq 0 \\ \text{sign}(b/a).flmax, & |b| > |a|.flmax \\ \text{sign}(b).flmax, & b \neq 0, a = 0 \end{cases} \quad (6)$$

where $flmax = 1/flmin$ and $flmin$ is the small positive value returned by X02AMF. The values of c and s are then computed or reconstructed via t as

$$c = \begin{cases} 1/(1 + t^2)^{1/2}, & \sqrt{\text{eps}} \leq |t| \leq 1/\sqrt{\text{eps}} \\ 1, & |t| < \sqrt{\text{eps}} \\ 1/|t|, & |t| > 1/\sqrt{\text{eps}} \end{cases} \quad s = \begin{cases} c.t, & \sqrt{\text{eps}} \leq |t| \leq 1/\sqrt{\text{eps}} \\ t, & |t| < \sqrt{\text{eps}} \\ \text{sign } t, & |t| > 1/\sqrt{\text{eps}} \end{cases} \quad (7)$$

where eps is the **machine precision**. Note that c is always non-negative in this scheme and that the same expressions are used in the initial computation of c and s from a and b as in any subsequent recovery of c and s via t . This is the approach used by many of the NAG Library routines that require plane rotations. d is computed simply as

$$d = c.a + s.b.$$

You need not be too concerned with the above detail, since routines are provided for setting up, recovering and applying such rotations.

Another use of plane rotations is to choose c and s so that for given x , y and z

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x & y \\ y & z \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}. \quad (8)$$

In such an application the matrix R is often termed a **Jacobi rotation** matrix. The routine that generates a Jacobi rotation (F06BEF) first computes the tangent t and then computes c and s via t as described above for the Givens rotation.

2.2.2 Complex plane rotations

In the complex case a plane rotation matrix for the (i, j) plane, R_{ij} is a unitary matrix and, analogously to the real case, it is usual to choose R_{ij} so that

$$R = \begin{pmatrix} \bar{c} & \bar{s} \\ -s & c \end{pmatrix}, \quad |c|^2 + |s|^2 = 1, \quad (9)$$

where \bar{a} denotes the complex conjugate of a .

The BLAS (see Lawson *et al.* (1979)) do not contain a routine for the generation of complex rotations, and so the routines in Chapter F06 are all based upon computing c and s via $t = b/a$ in an analogous manner to the real case. R can be chosen to have either c real, or s real and there are routines for both cases.

When c is real then it is non-negative and the transformation

$$\begin{pmatrix} c & \bar{s} \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix} \quad (10)$$

is such that if a is real then d is also real.

When s is real then the transformation

$$\begin{pmatrix} \bar{c} & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix} \quad (11)$$

is such that if b is real then d is also real.

2.2.3 Elementary real (Householder) reflections

There are a number of routines in the chapter concerned with setting up and applying Householder transformations. This section discusses the real case and the next section looks at the complex case. For further background information see Golub and Van Loan (1996).

A real elementary reflector, P , is a matrix of the form

$$P = I - \mu uu^T, \quad \mu u^T u = 2, \quad (12)$$

where μ is a scalar and u is a vector, and P is both symmetric and orthogonal. In the routines in Chapter F06, u is expressed in the form

$$u = \begin{pmatrix} \zeta \\ z \end{pmatrix}, \quad \zeta \text{ a scalar} \quad (13)$$

because in many applications ζ and z are not contiguous elements. The usual use of elementary reflectors is to choose μ and u so that for given α and x

$$P \begin{pmatrix} \alpha \\ x \end{pmatrix} = \begin{pmatrix} \beta \\ 0 \end{pmatrix}, \quad \alpha \text{ and } \beta \text{ scalars.} \quad (14)$$

Such a transformation is often termed a **Householder transformation**. There are two choices of μ and u available in Chapter F06.

The first form of the Householder transformation is compatible with that used by LINPACK (see Dongarra *et al.* (1979)) and has

$$\mu = 1/\zeta. \quad (15)$$

This choice makes ζ satisfy

$$1 \leq \zeta \leq 2.$$

The second form, and the form used by many of the NAG Library routines, has

$$\mu = 1 \tag{16}$$

which makes

$$1 \leq \zeta \leq \sqrt{2}.$$

In both cases the special setting

$$\zeta = 0 \tag{17}$$

is used by the routines to flag the case where $P = I$.

Note that while there are routines to apply an elementary reflector to a vector, there are no routines available in Chapter F06 to apply an elementary reflector to a matrix. This is because such transformations can readily and efficiently be achieved by calls to the matrix-vector Level 2 BLAS routines. For example, to form PA for a given matrix

$$\begin{aligned} PA &= (I - \mu uu^T)A = A - \mu uu^T A \\ &= A - \mu ub^T, \quad b = A^T u, \end{aligned} \tag{18}$$

and so we can call a matrix-vector product routine to form $b = A^T u$ and then call a rank-one update routine to form $(A - \mu ub^T)$. Of course, we must skip the transformation when ζ has been set to zero.

2.2.4 Elementary complex (Householder) reflections

A complex elementary reflector, P , is a matrix of the form

$$P = I - \mu uu^H, \quad \mu u^H u = 2, \quad \mu \text{ real,}$$

where u^H denotes the complex conjugate of u^T , and P is both Hermitian and unitary. For convenience in a number of applications this definition can be generalized slightly by allowing μ to be complex and so defining the generalized elementary reflector as

$$P = I - \mu uu^H, \quad |\mu|^2 u^H u = \mu + \bar{\mu} \tag{19}$$

for which P is still unitary, but is no longer Hermitian.

The Chapter F06 routines choose μ and ζ so that

$$\text{Re}(\mu) = 1, \quad \text{Im}(\zeta) = 0 \tag{20}$$

and this reduces to (12) with the choice (16) when μ and u are real. This choice is used because μ and u can now be chosen so that in the Householder transformation (14) we can make

$$\text{Im}(\beta) = 0$$

and, as in the real case,

$$1 \leq \zeta \leq \sqrt{2}.$$

Rather than returning μ and ζ as separate parameters the Chapter F06 routines return the single complex value θ defined as

$$\theta = \zeta + i \cdot \text{Im}(\mu), \quad i = \sqrt{-1}.$$

Obviously ζ and μ can be recovered as

$$\zeta = \text{Re}(\theta), \quad \mu = 1 + i \cdot \text{Im}(\theta).$$

The special setting

$$\theta = 0$$

is used to flag the case where $P = I$, and

$$\operatorname{Re}(\theta) \leq 0, \quad \operatorname{Im}(\theta) \neq 0$$

is used to flag the case where

$$P = \begin{pmatrix} \gamma & 0 \\ 0 & I \end{pmatrix}, \quad \gamma \text{ a scalar} \quad (21)$$

and in this case θ actually contains the value of γ . Notice that with both (18) and (21) we merely have to supply $\bar{\theta}$ rather than θ in order to represent P^H .

3 Recommendations on Choice and Use of Available Routines

3.1 Naming Scheme

3.1.1 NAG names

Table 1 shows the naming scheme for the routines in this chapter.

		Level-0	Level-1	Level-2	Level-3
integer	Chapter F06 routine	–	F06D_F	–	–
‘real’	BLAS routine	F06A_F	F06E_F	F06P_F	F06Y_F
‘real’	Chapter F06 routine	F06B_F	F06F_F	F06Q_F	–
				F06R_F	
‘complex’	BLAS routine	–	F06G_F	F06S_F	F06Z_F
‘complex’	Chapter F06 routine	F06C_F	F06H_F	F06T_F	–
				F06U_F	
‘mixed type’	BLAS routine	–	F06J_F	–	–
‘mixed type’	Chapter F06 routine	–	F06K_F	F06V_F	–

Table 1

The heading ‘mixed type’ is for routines where a mixture of data types is involved, such as a routine that returns the real Euclidean length of a complex vector. In future marks of the Library, routines may be included in categories that are currently empty and further categories may be introduced.

3.1.2 BLAS names

Those routines which conform to the specifications of the BLAS may be called either by their NAG names or by their BLAS names.

In many implementations of the NAG Library, references to BLAS names may be linked to an efficient machine-specific implementation of the BLAS, usually provided by the vendor of the machine. Such implementations are stringently tested before being used with the NAG Library, to ensure that they correctly meet the specifications of the BLAS, and that they return the desired accuracy. Use of BLAS names is recommended for efficiency.

References to NAG routine names (beginning F06-) are always linked to the code provided in the NAG Library and may be significantly slower than the equivalent BLAS routine.

The names of the Level-2 and Level-3 BLAS follow a simple scheme (which is similar to that used for LAPACK routines in Chapters F07 and F08). Each name has the structure **XYZZZ**, where the components have the following meanings:

- the initial letter **X** indicates the data type (real or complex) and precision:
 - S real, single precision (in Fortran 77, REAL)
 - D real, double precision (in Fortran 77, DOUBLE PRECISION)
 - C complex, single precision (in Fortran 77, COMPLEX)
 - Z complex, double precision (in Fortran 77, COMPLEX*16 or DOUBLE COMPLEX)
- the second and third letters **YY** indicate the type of the matrix A (and in some cases its storage scheme):

GE general
 GB general band
 SY symmetric
 SP symmetric (packed storage)
 SB symmetric band
 HE (complex) Hermitian
 HP (complex) Hermitian (packed storage)
 HB (complex) Hermitian band
 TR triangular
 TP triangular (packed storage)
 TB triangular band

– the remaining 1, 2 or 3 letters **ZZZ** indicate the computation performed:

MV matrix-vector product
 MM matrix-matrix product
 R rank-1 update
 R2 rank-2 update
 RK rank- k update
 R2K rank- $2k$ update
 SV solve a system of linear equations
 SM solve a system of linear equations with a matrix of right-hand sides

Thus the routine DGEMV performs a matrix-vector product involving a real general matrix in double precision; the corresponding routine for a complex general matrix is ZGEMV.

The names of the Level-1 BLAS mostly follow the same convention for the initial letter (S-, C-, D- or Z-), except for a few involving data of mixed type, where the first two characters are precision-dependent.

3.2 The Level-0 Scalar Routines

The Level-0 routines perform operations on scalars or on vectors or matrices of order 2.

3.3 The Level-1 Vector Routines

The Level-1 routines perform operations either on a single vector or on a pair of vectors.

3.4 The Level-2 Matrix-vector and Matrix Routines

The Level-2 routines perform operations involving either a matrix on its own, or a matrix and one or more vectors.

3.5 The Level-3 Matrix-matrix Routines

The Level-3 routines perform operations involving matrix-matrix products.

3.6 Vector Arguments

Vector arguments (except in the Level-1 Sparse BLAS) are represented by a one-dimensional array, immediately followed by an **increment** argument whose name consists of the three characters INC followed by the name of the array. For example, a vector x is represented by the two arguments X and INCX. The length of the vector, n say, is passed as a separate argument, N.

The increment argument is the spacing (stride) in the array between the elements of the vector. For instance, if $\text{INCX} = 2$, then the elements of x are in locations $x(1), x(3), \dots, x(2n - 1)$ of the array X and the intermediate locations $x(2), x(4), \dots, x(2n - 2)$ are not referenced.

When $\text{INCX} > 0$, the vector element x_i is in the array element $X(1 + (i - 1) \times \text{INCX})$. When $\text{INCX} \leq 0$, the elements are stored in the reverse order so that the vector element x_i is in the array element $X(1 - (n - i) \times \text{INCX})$ and hence, in particular, the element x_n is in X(1). The declared length of the array X in the calling subroutine must be at least $(1 + (N - 1) \times |\text{INCX}|)$.

Negative increments are permitted only for:

Level-1 routines which have more than one vector argument;

Level-2 BLAS routines (but not for other Level-2 routines)

Zero increments are formally permitted for Level-1 routines with more than one argument (in which case the element $X(1)$ is accessed repeatedly), but their use is strongly discouraged since the effect may be implementation-dependent. There is usually an alternative routine in this chapter, with a simplified parameter list, to achieve the required purpose. Zero increments are not permitted in the Level-2 BLAS.

In the Level-1 Sparse BLAS, each routine operates on two vectors x and y . The vector x is stored as a compressed sparse vector, and is represented by the three arguments NZ, X and INDX; NZ is the number of ‘interesting’ (usually nonzero) elements of x , and INDX is a one-dimensional **index** array such that

$$x(\text{INDX}(k)) = X(k), \quad k = 1, 2, \dots, \text{NZ}.$$

The (mathematical) length of the vector, n say, does not need to be supplied; it is assumed that $1 \leq \text{INDX}(k) \leq n$. For example, the vector

$$x = (0, 4, 0, 0, 1, 0, 0, 0, 6, 0)$$

could be represented with $\text{NZ} = 3$, $X = (4, 1, 6)$, $\text{INDX} = (2, 5, 9)$. The second vector y is stored conventionally, and is represented simply by the one-dimensional array Y, with y_i in $Y(i)$; the increment is assumed to be 1. Only the elements $Y(\text{INDX}(k))$ are referenced.

Non-positive values of NZ are permitted, in which case the routines return immediately — except that functions set their value to zero before returning. For those routines where Y is an output argument **the values in the array INDX must be distinct**; violating this condition may yield incorrect results.

3.7 Matrix Arguments and Storage Schemes

In this chapter the following different storage schemes are used for matrices:

- **conventional storage** in a two-dimensional array;
- **packed storage** for symmetric, Hermitian or triangular matrices;
- **band storage** for band matrices.

These storage schemes are compatible with those used in Chapters F07 and F08. (Different schemes for packed or band storage are used in a few older routines in Chapters F01, F02, F03 and F04.)

Chapter F01 provides some utility routines for conversion between storage schemes.

In the examples below, * indicates an array element which need not be set and is not referenced by the routines. The examples illustrate only the relevant leading rows and columns of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran 77.

3.7.1 Conventional storage

The default scheme for storing matrices is the obvious one: a general m by n matrix A is stored in a two-dimensional array A, with matrix element a_{ij} stored in array element $A(i, j)$. In the argument-list, the array name A is followed by its leading dimension LDA, as declared in the calling subroutine.

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

	Triangular matrix A	Storage in array A
UPLO = 'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$

UPLO = 'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$
------------	--	--

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 = 'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
UPLO = 'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

3.7.2 Packed storage

Symmetric, Hermitian or triangular matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed by columns in a one-dimensional array. In this chapter, as in Chapters F07 and F08, arrays which hold matrices in packed storage, have names ending in P. For a matrix of order n , the array must have at least $n(n + 1)/2$ elements. So:

if UPLO = 'U', a_{ij} is stored in $AP(i + j(j - 1)/2)$ for $i \leq j$;

if UPLO = 'L', a_{ij} is stored in $AP(i + (2n - j)(j - 1)/2)$ for $j \leq i$.

For example:

	Triangle of matrix A	Packed storage in array AP
UPLO = 'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11} \mid a_{12}a_{22} \mid \underbrace{a_{13}a_{23}a_{33}} \mid \underbrace{a_{14}a_{24}a_{34}a_{44}}$
UPLO = 'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$a_{11}a_{21}a_{31}a_{41} \mid \underbrace{a_{22}a_{32}a_{42}} \mid \underbrace{a_{33}a_{43}} \mid a_{44}$

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

3.7.3 Band storage

A band matrix with k_l subdiagonals and k_u superdiagonals may be stored compactly in a two-dimensional array with $k_l + k_u + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $k_l, k_u \ll n$, although the routines in this chapter, like those in Chapters F07 and F08, work correctly for all values of k_l and k_u .

To be precise, a_{ij} is stored in $A(k_u + 1 + i - j, j)$ for $\max(1, j - k_u) \leq i \leq \min(n, j + k_l)$. For example, when $n = 5$, $k_l = 2$ and $k_u = 1$:

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

Triangular band matrices are stored in the same format, with either $k_l = 0$ if upper triangular, or $k_u = 0$ 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:

if UPLO = 'U', a_{ij} is stored in $A(k + 1 + i - j, j)$ for $\max(1, j - k) \leq i \leq j$;

if UPLO = 'L', a_{ij} is stored in $A(1 + i - j, j)$ for $j \leq i \leq \min(n, j + k)$.

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

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

Note that different storage schemes for band matrices are used by some older routines in Chapters F01, F02, F03 and F04.

3.7.4 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 = 'U' (Unit triangular), 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.

3.7.5 Real diagonal elements of complex Hermitian matrices

Complex Hermitian matrices have diagonal elements that are by definition purely real. If such matrices are supplied as input to routines in this chapter, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by the routines, the computed imaginary parts are explicitly set to zero.

3.7.6 Spiked matrices

A few routines in this chapter (F06QSF, F06QWF, F06TSF and F06TWF) deal with **upper spiked** matrices. These are upper triangular matrices with an additional nonzero row or column below the diagonal.

The position of the spike is defined by indices k_1 and k_2 ; it is assumed that $k_1 < k_2$. A **row spike** has nonzero elements in the k_2 th row, $a_{k_2,k}$ for $k = k_1, k_1 + 1, \dots, k_2 - 1$; a **column spike** has nonzero elements in the k_1 th column, a_{k+1,k_1} for $k = k_1, k_1 + 1, \dots, k_2 - 1$. For example, when $n = 6$, $k_1 = 2$ and $k_2 = 5$:

Row spike	Column spike
$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ & & a_{33} & a_{34} & a_{35} & a_{36} \\ & & & a_{44} & a_{45} & a_{46} \\ & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ & & & & & a_{66} \end{pmatrix}$	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ & & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ & & & a_{42} & & a_{44} & a_{45} & a_{46} \\ & & & & a_{52} & & a_{55} & a_{56} \\ & & & & & & & a_{66} \end{pmatrix}$

The storage scheme adopted by the routines in this chapter is for the upper triangular part of the spiked matrix to be stored conventionally in a two-dimensional array A, with the subdiagonal elements of the spike stored in a separate vector.

3.8 Option Parameters

Many of the routines in this chapter have one or more **option parameters**, of type CHARACTER. The descriptions in the routine documents refer only to upper-case values (for example UPLO = 'U' or UPLO = 'L'); however, in every case, the corresponding lower-case characters may be supplied (with the same meaning). Any other value is illegal.

A longer character string can be passed as the actual parameter, making the calling program more readable, but only the first character is significant. (This is a feature of Fortran 77.) For example:

```
CALL DTRSV('Upper', 'Transpose', 'Non-unit', ...)
```

The following option arguments are used in this chapter:

- If TRANS = 'N', operate with the matrix (Not transposed);
- if TRANS = 'T', operate with the Transpose of the matrix;
- if TRANS = 'C', operate with the Conjugate transpose of the matrix.

- If UPLO = 'U', upper triangle or trapezoid of matrix;
- if UPLO = 'L', lower triangle or trapezoid of matrix.

- If DIAG = 'U', unit triangular;
- if DIAG = 'N', nonunit triangular.

- If SIDE = 'L', operate from the left-hand side;
- if SIDE = 'R', operate from the right-hand side.

- If PIVOT = 'V', variable pivot (in applying a sequence of plane rotations);
- if PIVOT = 'B', bottom pivot;
- if PIVOT = 'T', top pivot;
- if PIVOT = 'F', fixed pivot.

- If DIRECT = 'B', backward sequence of plane rotations;
- if DIRECT = 'F', forward sequence of plane rotations.

- If NORM = '1' or 'O', 1-norm of a matrix;
- if NORM = 'I', ∞ -norm of a matrix;
- if NORM = 'F' or 'E', Frobenius or Euclidean norm of a matrix;
- if NORM = 'M', maximum absolute value of the elements of a matrix (not strictly a norm).

- If MATRIX = 'G', general (rectangular or square) matrix;
- if MATRIX = 'U', upper trapezoidal or triangular matrix;
- if MATRIX = 'L', lower trapezoidal or triangular matrix.

3.8.1 Matrix norms

The option argument NORM specifies different matrix norms whose definitions are given here for reference (for a general m by n matrix A):

One-norm (NORM = 'O' or '1'):

$$\|A\|_1 = \max_j \sum_{i=1}^m |a_{ij}|;$$

Infinity-norm (NORM = 'I'):

$$\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|;$$

Frobenius or Euclidean norm (NORM = 'F' or 'E'):

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}.$$

If A is symmetric or Hermitian, $\|A\|_1 = \|A\|_\infty$.

The argument NORM can also be used to specify the maximum absolute value $\max_{i,j} |a_{ij}|$ (if NORM = 'M'), but this is not a norm in the strict mathematical sense.

3.9 Error Handling

Routines in this chapter do not use the usual NAG Library error-handling mechanism, involving the parameter IFAIL.

If one of the Level-2 or Level-3 BLAS routines is called with an invalid value of one of its arguments, then an error message is output on the error message unit (see X04AAF), giving the name of the routine and the number of the first invalid argument, and execution of the program is terminated. The following values of arguments are invalid:

- any value of the character arguments TRANS, TRANSA, TRANSB, UPLO, SIDE or DIAG, whose meaning is not specified;
- a negative value of any of the arguments M, N, K, KL or KU;
- too small a value for any of the leading dimension arguments;
- a zero value for the increment arguments INCX and INCY.

Zero values for the matrix dimensions M, N or K are considered valid.

The other routines in this chapter do not report any errors in their arguments. Normally, if called, for example, with an unspecified value for one of the option arguments, or with a negative value of one of the problem dimensions M or N, they simply do nothing and return immediately.

4 Index

Level 0 (Scalar) operations:

Complex numbers:

apply similarity rotation to 2 by 2 Hermitian matrix	F06CHF
generate a plane rotation, storing the tangent, real cosine	F06CAF
generate a plane rotation, storing the tangent, real sine	F06CBF
quotient of two numbers, with overflow flag	F06CLF
recover cosine and sine from given tangent, real cosine	F06CCF
recover cosine and sine from given tangent, real sine	F06CDF

Real numbers:

apply similarity rotation to 2 by 2 symmetric matrix	F06BHF
compute $(a^2 + b^2)^{1/2}$	F06BNF
compute Euclidean norm from scaled form	F06BMF

eigenvalue of 2 by 2 symmetric matrix	F06BPF
generate a Jacobi plane rotation	F06BEF
generate a plane rotation.....	F06AAF (DROTG)
generate a plane rotation storing the tangent	F06BAF
quotient of two numbers, with overflow flag.....	F06BLF
recover cosine and sine from given tangent.....	F06BCF
Level 1 (Vector) operations:	
Complex vector(s),	
add scalar times a vector to another vector	F06GCF (ZAXPY)
apply a complex plane rotation.....	F06HPF
apply an elementary reflection to a vector	F06HTF
apply a real plane rotation.....	F06KPF
apply plane rotation	
real cosine, complex sine.....	F06HMF (ZROT)
broadcast a scalar into a vector.....	F06HBF
copy a real vector to a complex vector.....	F06KFF
copy a vector.....	F06GFF (ZCOPY)
dot product of two vectors, conjugated	F06GBF (ZDOTC)
dot product of two vectors, unconjugated	F06GAF (ZDOTU)
Euclidean norm of a vector	F06JF (DZNRM2)
generate an elementary reflection	F06HRF
generate a sequence of plane rotations	F06HQF
index of element of largest absolute value.....	F06JMF (IZAMAX)
multiply vector by a complex scalar.....	F06GDF (ZSCAL)
multiply vector by a complex scalar, preserving input vector	F06HDF
multiply vector by a real scalar	F06JDF (ZDSCAL)
multiply vector by a real scalar, preserving input vector	F06KDF
multiply vector by complex diagonal matrix.....	F06HCF
multiply vector by real diagonal matrix	F06KCF
multiply vector by reciprocal of a real scalar	F06KEF
negate a vector	F06HGF
sum of absolute values of vector-elements.....	F06JKF (DZASUM)
swap two vectors.....	F06GGF (ZSWAP)
update Euclidean norm in scaled form.....	F06KJF
Integer vector(s),	
broadcast a scalar into a vector.....	F06DBF
copy a vector.....	F06DFB
Real vector(s),	
add scalar times a vector to another vector	F06ECF (DAXPY)
apply an elementary reflection to a vector (Linpack style).....	F06FUF
apply an elementary reflection to a vector (NAG style).....	F06FTF
apply a symmetric plane rotation to two vectors.....	F06FPF
apply plane rotation.....	F06EPF (DROT)
broadcast a scalar into a vector.....	F06FBF
copy a vector.....	F06EFF (DCOPY)
cosine of angle between two vectors.....	F06FAF
dot product of two vectors	F06EAF (DDOT)
elements of largest and smallest absolute value.....	F06FLF
Euclidean norm of a vector	F06EJF (DNRM2)
generate an elementary reflection (Linpack style).....	F06FSF
generate an elementary reflection (NAG style)	F06FRF
generate a sequence of plane rotations	F06FQF
index of element of largest absolute value.....	F06JLF (IDAMAX)
index of last non-negligible element.....	F06KLF
multiply vector by a scalar.....	F06EDF (DSCAL)
multiply vector by a scalar, preserving input vector	F06FDF
multiply vector by diagonal matrix.....	F06FCF
multiply vector by reciprocal of a scalar.....	F06FEF

negate a vector	F06FGF
sum of absolute values of vector-elements.....	F06EKF (DASUM)
swap two vectors.....	F06EGF (DSWAP)
update Euclidean norm in scaled form.....	F06FJF
weighted Euclidean norm of a vector.....	F06FKF
Level 2 (Matrix-vector and matrix) operations:	
Complex matrix and vector(s),	
apply sequence of plane rotations to a rectangular matrix:	
complex cosine, real sine.....	F06TYF
real cosine, complex sine.....	F06TXF
real cosine and sine.....	F06VXF
compute a norm or the element of largest absolute value:	
band matrix.....	F06UBF
general matrix.....	F06UAF
Hermitian band matrix.....	F06UEF
Hermitian matrix.....	F06UCF
Hermitian matrix, packed form.....	F06UDF
Hermitian tridiagonal matrix.....	F06UPF
Hessenberg matrix.....	F06UMF
symmetric band matrix.....	F06UHF
symmetric matrix.....	F06UFF
symmetric matrix, packed form.....	F06UGF
trapezoidal matrix.....	F06UJF
triangular band matrix.....	F06ULF
triangular matrix, packed form.....	F06UKF
tridiagonal matrix.....	F06UNF
compute upper Hessenberg matrix by applying sequence of plane rotations to an upper triangular matrix.....	F06TVF
compute upper spiked matrix by applying sequence of plane rotations to an upper triangular matrix.....	F06TWF
matrix initialization.....	F06THF
matrix-vector product,	
Hermitian band matrix.....	F06SDF (ZHBMV)
Hermitian matrix.....	F06SCF (ZHEMV)
Hermitian packed matrix.....	F06SEF (ZHPMV)
rectangular band matrix.....	F06SBF (ZGBMV)
rectangular matrix.....	F06SAF (ZGEMV)
symmetric matrix.....	F06TAF
symmetric packed matrix.....	F06TCF
triangular band matrix.....	F06SGF (ZTBMV)
triangular matrix.....	F06SFF (ZTRMV)
triangular packed matrix.....	F06SHF (ZTPMV)
permute rows or columns of a matrix:	
permutations represented by an integer array.....	F06VJF
permutations represented by a real array.....	F06VKF
QR factorization by sequence of plane rotations:	
of rank-1 update of upper triangular matrix.....	F06TPF
of upper triangular matrix augmented by a full row.....	F06TQF
QR factorization of UZ or RQ factorization of ZU , where U is upper triangular and Z is a sequence of plane rotations.....	F06TTF
QR or RQ factorization by sequence of plane rotations:	
of upper Hessenberg matrix.....	F06TRF
of upper spiked matrix.....	F06TSF
rank-1 update,	
Hermitian matrix.....	F06SPF (ZHER)
Hermitian packed matrix.....	F06SQF (ZHPR)
rectangular matrix, conjugated vector.....	F06SNF (ZGERC)
rectangular matrix, unconjugated vector.....	F06SMF (ZGERU)

symmetric matrix.....	F06TBF
symmetric packed matrix.....	F06TDF
rank-2 update,	
Hermitian matrix.....	F06SRF (ZHER2)
Hermitian packed matrix.....	F06SSF (ZHPR2)
matrix copy, rectangular or trapezoidal.....	F06TFF
solution of a system of equations:	
triangular band matrix.....	F06SKF (ZTBSV)
triangular matrix.....	F06SJF (ZTRSV)
triangular packed matrix.....	F06SLF (ZTPSV)
unitary similarity transformation of a Hermitian matrix, as sequence of plane rotations.....	F06TMF
Real matrix and vector(s),	
apply sequence of plane rotations to a rectangular matrix.....	F06QXF
compute a norm or the element of largest absolute value:	
band matrix.....	F06RBF
general matrix.....	F06RAF
Hessenberg matrix.....	F06RMF
matrix initialization.....	F06QHF
symmetric band matrix.....	F06REF
symmetric matrix.....	F06RCF
symmetric matrix, packed form.....	F06RDF
symmetric tridiagonal matrix.....	F06RPF
trapezoidal matrix.....	F06RJF
triangular band matrix.....	F06RLF
triangular matrix, packed form.....	F06RKF
tridiagonal matrix.....	F06RNF
compute upper Hessenberg matrix by applying sequence of plane rotations to an upper triangular matrix.....	F06QVF
compute upper spiked matrix by applying sequence of plane rotations to an upper triangular matrix.....	F06QWF
matrix-vector product,	
rectangular band matrix.....	F06PBF (DGBMV)
rectangular matrix.....	F06PAF (DGEMV)
symmetric band matrix.....	F06PDF (DSBMV)
symmetric matrix.....	F06PCF (DSYMV)
symmetric packed matrix.....	F06PEF (DSPMV)
triangular band matrix.....	F06PGF (DTBMV)
triangular matrix.....	F06PFF (DTRMV)
triangular packed matrix.....	F06PHF (DTPMV)
orthogonal similarity transformation of a symmetric matrix, as sequence of plane rotations.....	F06QMF
permute rows or columns of a matrix:	
permutations represented by an integer array.....	F06QJF
permutations represented by a real array.....	F06QKF
QR factorization by sequence of plane rotations:	
of rank-1 update of upper triangular matrix.....	F06QPF
of upper triangular matrix augmented by a full row.....	F06QQF
QR factorization of UZ or RQ factorization of ZU , where U is upper triangular and Z is a sequence of plane rotations.....	F06QTF
QR or RQ factorization by sequence of plane rotations:	
of upper Hessenberg matrix.....	F06QRF
of upper spiked matrix.....	F06QSF
rank-1 update,	
rectangular matrix.....	F06PMF (DGER)
symmetric matrix.....	F06PPF (DSYR)
symmetric packed matrix.....	F06PQF (DSPR)
rank-2 update,	
matrix copy, rectangular or trapezoidal.....	F06QFF

symmetric matrix	F06PRF (DSYR2)
symmetric packed matrix	F06PSF (DSPR2)
solution of system of equations, triangular band matrix	F06PKF (DTBSV)
triangular matrix	F06PJF (DTRSV)
triangular packed matrix	F06PLF (DTPSV)

Level 3 (Matrix-matrix) operations:

Complex matrices:

matrix-matrix product:	
one matrix Hermitian	F06ZCF (ZHEMM)
one matrix symmetric.....	F06ZTF (ZSYMM)
triangular matrix	F06ZFF (ZTRMM)
two rectangular matrices	F06ZAF (ZGEMM)
rank-2k update:	
of a Hermitian matrix	F06ZRF (ZHER2K)
of a symmetric matrix.....	F06ZWF (ZSYR2K)
rank-k update:	
of a Hermitian matrix	F06ZPF (ZHERK)
of a symmetric matrix.....	F06ZUF (ZSYRK)
solution of triangular systems of equations	F06ZJF (ZTRSM)

Real matrices:

matrix-matrix product:	
one matrix symmetric.....	F06YCF (DSYMM)
one matrix triangular.....	F06YFF (DTRMM)
rectangular matrices.....	F06YAF (DGEMM)
rank-2k update of a symmetric matrix.....	F06YRF (DSYR2K)
rank-k update of a symmetric matrix.....	F06YPF (DSYRK)
solution of triangular systems of equations	F06YJF (DTRSM)

Sparse level 1 (vector) operations:

Complex vectors:

add scalar times sparse vector to another sparse vector.....	F06GTF (ZAXPYI)
dot product of two sparse vectors (conjugated).....	F06GSF (ZDOTCI)
dot product of two sparse vectors (unconjugated).....	F06GRF (ZDOTUI)
gather and set to zero a sparse vector.....	F06GVF (ZGTHRZ)
gather sparse vector.....	F06GUF (ZGTHR)
scatter sparse vector	F06GWF (ZSCTR)

Real vectors:

add scalar times sparse vector to another sparse vector.....	F06ETF (DAXPYI)
apply plane rotation to two sparse vectors.....	F06EXF (DROTI)
dot product of two sparse vectors.....	F06ERF (DDOTI)
gather and set to zero a sparse vector.....	F06EVF (DGTHRZ)
gather sparse vector.....	F06EUF (DGTHR)
scatter sparse vector	F06EWF (DSCTR)

BLAS Routines

Real Matrices			Complex Matrices		
single precision	double precision	NAG	single precision	double precision	NAG
ISAMAX	IDAMAX	F06JLF	ICAMAX	IZAMAX	F06JMF
SASUM	DASUM	F06EKF	CAXPY	ZAXPY	F06GCF
SAXPY	DAXPY	F06ECF	CAXPYI	ZAXPYI	F06GTF
SAXPYI	DAXPYI	F06ETF	CCOPY	ZCOPY	F06GFF
SCASUM	DZASUM	F06JKF	CDOTC	ZDOTC	F06GBF
SCNRM2	DZNRM2	F06JFF	CDOTCI	ZDOTCI	F06GSF
SCOPY	DCOPY	F06EFF	CDOTU	ZDOTU	F06GAF
SDOT	DDOT	F06EAF	CDOTUI	ZDOTUI	F06GRF

SDOTI	DDOTI	F06ERF	CGBMV	ZGBMV	F06SBF
SGBMV	DGBMV	F06PBF	CGEMM	ZGEMM	F06ZAF
SGEMM	DGEMM	F06YAF	CGEMV	ZGEMV	F06SAF
SGEMV	DGEMV	F06PAF	CGERC	ZGERC	F06SNF
SGER	DGER	F06PMF	CGERU	ZGERU	F06SMF
SGTHR	DGTHR	F06EUF	CGTHR	ZGTHR	F06GUF
SGTHRZ	DGTHRZ	F06EVF	CGTHRZ	ZGTHRZ	F06GVF
SNRM2	DNRM2	F06EJF	CHBMV	ZHBMV	F06SDF
SROT	DROT	F06EPF	CHEMM	ZHEMM	F06ZCF
SROTG	DROTG	F06AAF	CHEMV	ZHEMV	F06SCF
SROTI	DROTI	F06EXF	CHER	ZHER	F06SPF
SSBMV	DSBMV	F06PDF	CHER2	ZHER2	F06SRF
SSCAL	DSCAL	F06EDF	CHER2K	ZHER2K	F06ZRF
SSCTR	DSCTR	F06EWF	CHERK	ZHERK	F06ZPF
SSPMV	DSPMV	F06PEF	CHPMV	ZHPMV	F06SEF
SSPR	DSPR	F06PQF	CHPR	ZHPR	F06SQF
SSPR2	DSPR2	F06PSF	CHPR2	ZHPR2	F06SSF
SSWAP	DSWAP	F06EGF	CSCAL	ZSCAL	F06GDF
SSYMM	DSYMM	F06YCF	CCTR	ZCTR	F06GWF
SSYMV	DSYMV	F06PCF	CSSCAL	ZDSCAL	F06JDF
SSYR	DSYR	F06PPF	CSWAP	ZSWAP	F06GGF
SSYR2	DSYR2	F06PRF	CSYMM	ZSYMM	F06ZTF
SSYR2K	DSYR2K	F06YRF	CSYR2K	ZSYR2K	F06ZWF
SSYRK	DSYRK	F06YPF	CSYRK	ZSYRK	F06ZUF
STBMV	DTBMV	F06PGF	CTBMV	ZTBMV	F06SGF
STBSV	DTBSV	F06PKF	CTBSV	ZTBSV	F06SKF
STPMV	DTPMV	F06PHF	CTPMV	ZTPMV	F06SHF
STPSV	DTPSV	F06PLF	CTPSV	ZTPSV	F06SLF
STRMM	DTRMM	F06YFF	CTRMM	ZTRMM	F06ZFF
STRMV	DTRMV	F06PFF	CTRMV	ZTRMV	F06SFF
STRSM	DTRSM	F06YJF	CTRSM	ZTRSM	F06ZJF
STRSV	DTRSV	F06PJF	CTRSV	ZTRSV	F06SJF

5 Routines Withdrawn or Scheduled for Withdrawal

None.

6 References

- Dodson D S and Grimes R G (1982) Remark on Algorithm 539 *ACM Trans. Math. Software* **8** 403–404
- Dodson D S, Grimes R G and Lewis J G (1991) Sparse extensions to the Fortran basic linear algebra subprograms *ACM Trans. Math. Software* **17** 253–263
- 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
- Dongarra J J, Moler C B, Bunch J R and Stewart G W (1979) *LINPACK Users' Guide* SIAM, Philadelphia

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Lawson C L, Hanson R J, Kincaid D R and Krogh F T (1979) Basic linear algebra subprograms for Fortran usage *ACM Trans. Math. Software* **5** 308–325
