An Updated Set of Basic Linear Algebra Subprograms (BLAS)

L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling,
G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet,
R. Pozo, K. Remington, R. C. Whaley

January 25, 2001

1 Introduction

Numerical linear algebra, particularly the solution of linear systems of equations, linear least squares problems, eigenvalue problems and singular value problems, is fundamental to most calculations in scientific computing, and is often the most computationally intensive part of such calculations. Designers of computer programs involving linear algebraic operations have frequently chosen to implement certain low level operations, such as the dot product or the matrix-vector product, as separate subprograms. This may be observed both in many published codes and in codes written for specific applications at many computer installations.

This approach encourages structured programming and improves the self-documenting quality of the software by specifying basic building blocks and identifying these operations with unique mnemonic names. Since a significant amount of execution time in complicated linear algebraic programs may be spent in a few low level operations, reducing the execution time spent in these operations leads to an overall reduction in the execution time of the program. The programming of some of these low level operations involves algorithmic and implementation subtleties that need care, and can be easily overlooked. If there is general agreement on standard names and parameter lists for some of these basic operations, then portability and efficiency can also be achieved.

This paper summarizes the BLAS Technical Forum Standard, a specification of a set of kernel routines for linear algebra, historically called the Basic Linear Algebra Subprograms and commonly known as the BLAS. The complete standard can be found in [1], and on the BLAS Technical Forum webpage (`http://www.netlib.org/blas/blast-forum/`).

The first major concerted effort to achieve agreement on the specification of a set of linear algebra kernels resulted in the Level 1 Basic Linear Algebra Subprograms (BLAS)[1] [18] and associated test suite. The Level 1 BLAS are the specification and implementation in Fortran of subprograms for scalar and vector operations. This was the result of a collaborative project in 1973-77. Following the distribution of the initial version of the specifications to people active in the development of numerical linear algebra software, a series of open meetings were held at conferences and, as a result, extensive modifications were made in an effort to improve the design and make the subprograms more robust. The Level 1 BLAS were extensively and success-

[1]Originally known just as the BLAS, but in the light of subsequent developments now known as the Level 1 BLAS

fully exploited by LINPACK [9], a software package for the solution of dense and banded linear equations and linear least squares problems.

With the advent of vector machines, hierarchical memory machines and shared memory parallel machines, specifications for the Level 2 and 3 BLAS [11, 10], concerned with matrix-vector and matrix-matrix operations respectively, were drawn up in 1984-86 and 1987-88. These specifications made it possible to construct new software to utilize the memory hierarchy of modern computers more effectively. In particular, the Level 3 BLAS allowed the construction of software based upon block-partitioned algorithms, typified by the linear algebra software package LAPACK [2]. LAPACK is state-of-the-art software for the solution of dense and banded linear equations, linear least squares, eigenvalue and singular value problems. LAPACK makes extensive use of all levels of BLAS and particularly utilizes the Level 2 and 3 BLAS for portable performance. LAPACK is widely used in application software and is supported by a number of hardware and software vendors.

To a great extent, the user community embraced the BLAS, not only for performance reasons, but also because developing software around a core of common routines like the BLAS is good software engineering practice. Highly efficient machine-specific implementations of the BLAS are available for most modern high-performance computers. The BLAS have enabled software to achieve high performance with portable code.

The original BLAS concentrated on dense and banded operations, but many applications require the solution of problems involving sparse matrices, and thus there have also been efforts to specify computational kernels for sparse vector and matrix operations [8, 12].

In the spirit of the earlier BLAS meetings and the standardization efforts of the MPI and HPF forums, a technical forum was established to consider expanding the BLAS in the light of modern software, language, and hardware developments. The BLAS Technical Forum meetings began with a workshop in November 1995 at the University of Tennessee. Meetings were hosted by universities, government institutions, and software and hardware vendors. Detailed minutes were taken for each of the meetings, and these minutes are available on the BLAS Technical Forum webpage (`http://www.netlib.org/blas/blast-forum/`).

Various working groups within the Technical Forum were established to consider issues such as the overall functionality, language interfaces, sparse BLAS, distributed-memory dense BLAS, extended and mixed precision BLAS, interval BLAS, and extensions to the existing BLAS. The rules of the forum were adopted from those used for the MPI and HPF forums. In other words, final acceptance of each of the chapters in the BLAS Technical Forum standard were decided at the meetings using *Robert's Rules*. Drafts of the document were also available on the BLAS Technical Forum webpage, and attendees were permitted to edit chapters, give comments, and vote on-line in "virtual meetings", as well as to conduct discussions on the email reflector. The results of these working groups are summarized in this paper. Most of these discussions resulted in definitive proposals which led to the specifications given in [5, 14, 16]. Not all of the discussions resulted in definitive proposals, and such discussions are summarized in what was called the "Journal of Development" in the hope that they may encourage future efforts to take those discussions to a successful conclusion. The "Journal of Development" forms an appendix to the standard (see `http://www.netlib.org/blas/blast-forum/`).

A major aim of the standards defined in the document is to enable linear algebra libraries (both public domain and commercial) to interoperate efficiently, reliably and easily. We believe that hardware and software vendors, higher level library writers and application programmers all benefit from the efforts of this forum and are the intended end users of these standards.

The specification of the original BLAS was given in the form of Fortran 66 and subsequently Fortran 77 subprograms. In the BLAS Technical Forum standard, we provide specifications for Fortran 95[2], Fortran 77 and C. Reference implementations of the standard are provided on the BLAS Technical Forum webpage (`http://www.netlib.org/blas/blast-forum/`). Alternative language bindings for C++ and Java were also discussed during the meetings of the forum, but the specifications for these bindings were postponed for a future series of meetings.

The remainder of this paper is organized as follows. Section 2 provides motivation for the functionality, and Sections 3 and 4 define mathematical notation and present tables of functionality for the routines. Section 5 discusses the naming scheme for the routines. Section 6 discusses issues concerning the numerical accuracy of the BLAS, and Section 7 briefly details the error handling mechanisms utilized within the routines. And lastly, Section 8 acknowledges all of the individuals who have contributed to this standardization effort.

2 Motivation

The motivation for the kernel operations is proven functionality. Many of the new operations are based upon auxiliary routines in LAPACK [2] (e.g.,

[2]the current Fortran standard

SUMSQ, GEN_GROT, GEN_HOUSE, SORT, GE_NORM, GE_COPY). Only after the LAPACK project was begun was it realized that there were operations like the matrix copy routine (GE_COPY), the computation of a norm of a matrix (GE_NORM) and the generation of Householder transformations (GEN_HOUSE) that occurred so often that it was wise to make separate routines for them.

A second group of these operations extended the functionality of some of the existing BLAS (e.g., AXPBY, WAXPBY, GER, SYR/HER, SPR/HPR, SYR2/HER2, SPR2/HPR2). For example, the Level 3 BLAS for the rank k update of a symmetric matrix only allows a positive update, which means that it cannot be used for the reduction of a symmetric matrix to tridiagonal form (to facilitate the computation of the eigensystem of a symmetric matrix), or for the factorization of a symmetric indefinite matrix, or for a quasi-Newton update in an optimization routine.

Other extensions (e.g., AXPY_DOT, GE_SUM_MV, GEMVT, TRMVT, GEMVER) perform two Level 1 BLAS (or Level 2 BLAS) routine calls simultaneously to increase performance by reducing memory traffic.

One important feature of the new standard is the inclusion of sparse matrix computational routines. Because there are many formats commonly used to represent sparse matrices, the Level 2 and Level 3 Sparse BLAS routines utilize an abstract representation, or handle, rather than a fixed storage description (e.g. compressed row, or skyline storage). This handle-based representation allows one to write portable numerical algorithms using the Sparse BLAS, independent of the matrix storage implementation, and gives BLAS library developers the best opportunity for optimizing and fine-tuning their kernels for specific architectures or application domains.

The original Level 2 BLAS included, as an appendix, the specification of extended precision subprograms. With the widespread adoption of hardware supporting the IEEE extended arithmetic format [17], as well as other forms of extended precision arithmetic, together with the increased understanding of algorithms to successfully exploit such arithmetic, it was felt to be timely to include a complete specification for a set of extra precise BLAS.

3 Nomenclature and Conventions

This section addresses mathematical notation and definitions for the BLAS routines.

The following notation is used in the functionality tables.

- A, B, C – matrices

- D, D_L, D_R – diagonal matrices
- H – Householder matrix
- J – symmetric tridiagonal matrix (including 2×2 blocked diagonal)
- P – permutation matrix
- T – triangular matrix
- u, v, w, x, y, z – vectors
- $\bar{x}$ – specifies the conjugate of the complex vector x
- $incu$, $incv$, $incw$, $incx$, $incy$, $incz$ – stride between successive elements of the respective vector
- Greek letters - scalars (but not exclusively Greek letters)
- x_i - an element of a one-dimensional array
- $y|_x$ – refers to the elements of the dense vector y that have common indices with the sparse vector x.
- $\leftarrow$ – assignment statement
- $\leftrightarrow$ – swap (assignment) statement
- $||\cdot||_p$ – the p-norm of a vector or matrix

Additional notation for sparse matrices can be found in the Sparse BLAS chapter of the BLAS Technical Forum standard [1] (`http://www.netlib.org/blas/blast-forum/`), as well as [16].

For the mathematical formulation of the operations, as well as their algorithmic presentation, we have chosen to index the vector and matrix operands starting from zero. This decision was taken to simplify the presentation of the document but has no impact on the convention a particular language binding may choose.

3.1 Scalar Arguments

Many scalar arguments are used in the specifications of the BLAS routines. For example, the size of a vector or matrix operand is determined by the integer argument(s) `m` and/or `n`. Note that it is permissible to call the routines with `m` or `n` equal to zero, in which case the routine exits immediately without referencing its vector/matrix elements. Some routines return a displacement denoted by the integer argument k. The scaling of a vector or matrix is often denoted by the arguments alpha and beta.

3.2 Vector Operands

A n-length vector operand x is specified by two arguments – x and incx. x is an array that contains the entries of the n-length vector x. incx is the stride within x between two successive elements of the vector x.

Lowercase letters are used to denote a vector.

3.3 Vector Norms

There are a variety of ways to define the norm of a vector, in particular for vectors of complex numbers, several of which have been used in the existing Level 1 BLAS and in various LAPACK auxiliary routines. Our definitions include all of these in a systematic way.

Data Type	Name	Notation	Definition
Real	one-norm	$\|x\|_1$	$\sum_i \|x_i\|$
	two-norm	$\|x\|_2$	$\sqrt{\sum_i x_i^2}$
	infinity-norm	$\|x\|_\infty$	$\max_i \|x_i\|$
Complex	one-norm	$\|x\|_1$	$\sum_i \|x_i\|$
			$= \sum_i (Re(x_i)^2 + Im(x_i)^2)^{1/2}$
	real one-norm	$\|x\|_{1R}$	$\sum_i (\|Re(x_i)\| + \|Im(x_i)\|)$
	two-norm	$\|x\|_2$	$\sqrt{\sum_i \|x_i\|^2}$
			$= (\sum_i (Re(x_i)^2 + Im(x_i)^2))^{1/2}$
	infinity-norm	$\|x\|_\infty$	$\max_i \|x_i\|$
			$= \max_i (Re(x_i)^2 + Im(x_i)^2)^{1/2}$
	real infinity-norm	$\|x\|_{\infty R}$	$\max_i (\|Re(x_i)\| + \|Im(x_i)\|)$

Table 1: Vector Norms

Rationale. The reason for the two extra norms of complex vectors, the real one-norm and real infinity-norm, is to avoid the expense of up to n square roots, where n is the length of the vector x. The two-norm only requires one square root, so a real version is not needed. The infinity norm only requires one square root in principle, but this would require tests and branches, making it more complicated and slower than the real infinity-norm. When x is real, the one-norm and real one-norm are identical, as are the infinity-norm and real infinity-norm. We note that the Level 1 BLAS routine ICAMAX, which finds the largest entry of a complex vector, finds the largest value of $|Re(x_i)|+|Im(x_i)|$. (*End of rationale.*)

Computing the two-norm or Frobenius-norm of a vector is equivalent. However, this is not the case for computing matrix norms. For consistency of notation between vector and matrix norms, both norms are available.

3.4 Matrix Operands

A m-by-n matrix operand A is specified by the argument A. A is a language-dependent data structure containing the entries of the matrix operand A. The representation of the matrix entry $a_{i,j}$ in A is denoted by A(i,j) for all (i,j) in the interval $[0 \ldots m-1] \times [0 \ldots n-1]$.

Capital letters are used to denote a matrix.

3.5 Matrix Norms

Analogously to vector norms as discussed in Section 3.3, there are a variety of ways to define the norm of a matrix, in particular for matrices of complex numbers. Our definitions include all of these in a systematic way.

In contrast to computing vector norms, computing the two-norm and Frobenius-norm of a matrix are not equivalent. If the user asks for the two-norm of a matrix, where the matrix is 2-by-2 or larger, an error flag is raised. The one exception occurs when the matrix is a single column or a single row. In this case, the two-norm is requested and the Frobenius-norm is returned.

4 Functionality

This section summarizes, in tabular form, the functionality of the proposed routines. Issues such as storage formats or data types are not addressed. The functionality of the existing Level 1, 2 and 3 BLAS [8, 10, 11, 18] is a subset of the functionality proposed in this document.

In the original BLAS, each level was categorized by the type of operation; Level 1 addressed scalar and vector operations, Level 2 addressed matrix-vector operations, while Level 3 addressed matrix-matrix operations. The functionality tables in this document are categorized in a similar manner,

Data Type	Name	Notation	Definition
Real	one-norm	$\|A\|_1$	$\max_j \sum_i \lvert a_{ij}\rvert$
	Frobenius-norm	$\|A\|_F$	$\sqrt{\sum_i \sum_j a_{ij}^2}$
	infinity-norm	$\|A\|_\infty$	$\max_i \sum_j \lvert a_{ij}\rvert$
	max-norm	$\|A\|_{\max}$	$\max_i \max_j \lvert a_{ij}\rvert$
Complex	one-norm	$\|A\|_1$	$\max_j \sum_i \lvert a_{ij}\rvert$
			$= \max_j \sum_i (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real one-norm	$\|A\|_{1R}$	$\max_j \sum_i (\lvert Re(a_{ij})\rvert + \lvert Im(a_{ij})\rvert)$
	Frobenius-norm	$\|A\|_F$	$\sqrt{\sum_i \sum_j \lvert a_{ij}\rvert^2}$
			$= (\sum_i \sum_j (Re(a_{ij})^2 + Im(a_{ij})^2))^{1/2}$
	infinity-norm	$\|A\|_\infty$	$\max_i \sum_j \lvert a_{ij}\rvert$
			$= \max_i \sum_j (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real infinity-norm	$\|A\|_{\infty R}$	$\max_i \sum_j (\lvert Re(a_{ij})\rvert + \lvert Im(a_{ij})\rvert)$
	max-norm	$\|A\|_{\max}$	$\max_i \max_j \lvert a_{ij}\rvert$
			$= \max_i \max_j (Re(a_{ij})^2 + Im(a_{ij})^2)^{1/2}$
	real max-norm	$\|A\|_{\max R}$	$= \max_i \max_j (\lvert Re(a_{ij})\rvert + \lvert Im(a_{ij})\rvert)$

Table 2: Matrix Norms

with additional categories to cover operations which were not addressed in the original BLAS.

Unless otherwise specified, the operations apply to both real and complex arguments. For the sake of compactness the complex operators are omitted, so that whenever a transpose operation is given the conjugate transpose should also be assumed for the complex case.

The last column of each table denotes in which chapter of the BLAS Technical Forum standard the functionality occurs. Specifically,

- "D" denotes dense and banded BLAS
- "S" denotes sparse BLAS, and
- "E" denotes extended and mixed precision BLAS.

4.1 Scalar and Vector Operations

This section lists scalar and vector operations. The functionality tables are organized as follows. Table 3 lists the scalar and vector reduction operations, Table 4 lists the vector rotation operations, Table 5 lists the vector operations, and Table 6 lists those vector operations that involve only data movement.

For the Sparse BLAS, x is a compressed sparse vector and y is a dense vector.

For further details of vector norm notation, refer to Section 3.3.

Dot product	$r \leftarrow \beta r + \alpha x^T y$	D,E
	$r \leftarrow x^T y$	S
Vector norms	$r \leftarrow \|\|x\|\|_1$,	D
	$r \leftarrow \|\|x\|\|_{1R}$,	D
	$r \leftarrow \|\|x\|\|_2$,	D
	$r \leftarrow \|\|x\|\|_\infty$,	D
	$r \leftarrow \|\|x\|\|_{\infty R}$,	D
Sum	$r \leftarrow \sum_i x_i$	D,E
Min value & location	$k, x_k, ; k = \arg\min_i x_i$	D
Min abs value & location	$k, x_k, k = \arg\min_i(\|Re(x_i)\| + \|Im(x_i)\|)$	D
Max value & location	$k, x_k, ; k = \arg\max_i x_i$	D
Max abs value & location	$k, x_k, k = \arg\max_i(\|Re(x_i)\| + \|Im(x_i)\|)$	D
Sum of squares	$(scl, ssq) \leftarrow \sum x_i^2$,	D
	$ssq \cdot scl^2 = \sum x_i^2$	D

Table 3: Reduction Operations

Generate Givens rotation	$(c, s, r) \leftarrow \text{rot}(a, b)$	D
Generate Jacobi rotation	$(a, b, c, s) \leftarrow \text{jrot}(x, y, z)$	D
Generate Householder transform	$(\alpha, x, \tau) \leftarrow \text{house}(\alpha, x)$, $H = I - \alpha u u^T$	D

Table 4: Generate Transformations

Reciprocal Scale	$x \leftarrow x/\alpha$	D
Scaled vector accumulation	$y \leftarrow \alpha x + \beta y,$	D,E
	$y \leftarrow \alpha x + y$	S
Scaled vector addition	$w \leftarrow \alpha x + \beta y$	D,E
Combined axpy & dot product	$\begin{cases} \hat{w} \leftarrow w - \alpha v \\ r \leftarrow \hat{w}^T u \end{cases}$	D
Apply plane rotation	$\begin{pmatrix} x & y \end{pmatrix} \leftarrow \begin{pmatrix} x & y \end{pmatrix} R$	D

Table 5: Vector Operations

Copy	$y \leftarrow x$	D
Swap	$y \leftrightarrow x$	D
Sort vector	$x \leftarrow \mathrm{sort}(x)$	D
Sort vector & return index vector	$(p, x) \leftarrow \mathrm{sort}(x)$	D
Permute vector	$x \leftarrow Px$	D
Sparse gather	$x \leftarrow y\|_x$	S
Sparse gather and zero	$x \leftarrow y\|_x;\ y\|_x \leftarrow 0$	S
Sparse scatter	$y\|_x \leftarrow x$	S

Table 6: Data Movement with Vectors

4.2 Matrix-Vector Operations

This section lists matrix-vector operations in Table 7. The matrix arguments A, B and T are dense or banded or sparse. In addition, where appropriate, the matrix A can be symmetric (Hermitian) or triangular or general. The matrix T represents an upper or lower triangular matrix, which can be unit or non-unit triangular. For the Sparse BLAS, the matrix A is sparse, the matrix T is sparse triangular, and the vectors x and y are dense.

Matrix-vector product	$y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y$	D,E
	$x \leftarrow \alpha Tx,\ x \leftarrow \alpha T^T x$	D,E
	$y \leftarrow \alpha Ax + y,\ y \leftarrow \alpha A^T x + y$	S
Summed matrix-vector multiplies	$y \leftarrow \alpha Ax + \beta Bx$	D,E
Multiple matrix-vector multiplies	$\begin{cases} x \leftarrow T^T y \\ w \leftarrow Tz \end{cases}$	D
	$\begin{cases} x \leftarrow \beta A^T y + z \\ w \leftarrow \alpha Ax \end{cases}$	D
Multiple matrix-vector mults and low rank updates	$\begin{cases} \hat{A} \leftarrow A + u_1 v_1^T + u_2 v_2^T \\ x \leftarrow \beta \hat{A}^T y + z \\ w \leftarrow \alpha \hat{A} x \end{cases}$	D
Triangular solve	$x \leftarrow \alpha T^{-1} x,\ x \leftarrow \alpha T^{-T} x$	D,S,E
Rank one updates	$A \leftarrow \alpha x y^T + \beta A$	D
and symmetric ($A = A^T$)	$A \leftarrow \alpha x x^T + \beta A$	D
rank one & two updates	$A \leftarrow (\alpha x) y^T + y (\alpha x)^T + \beta A$	D

Table 7: Matrix-Vector Operations

4.3 Matrix Operations

This section lists a variety of matrix operations. The functionality tables are organized as follows. Table 8 lists single matrix operations and matrix operations that involve $O(n^2)$ operations, Table 9 lists the $O(n^3)$ matrix-matrix operations and Table 10 lists those matrix operations that involve only data movement. Where appropriate one or more of the matrices can also be symmetric (Hermitian) or triangular or general. The matrix T represents an upper or lower triangular matrix, which can be unit or non-unit triangular. D, D_L, and D_R represent diagonal matrices, and J represents a symmetric tridiagonal matrix (including 2×2 block diagonal).

For further details of matrix norm notation, refer to Section 3.5.

Matrix norms	$r \leftarrow \|\|A\|\|_1, r \leftarrow \|\|A\|\|_{1R}$	D
	$r \leftarrow \|\|A\|\|_F, r \leftarrow \|\|A\|\|_\infty, r \leftarrow \|\|A\|\|_{\infty R}$	D
	$r \leftarrow \|\|A\|\|_{max}, r \leftarrow \|\|A\|\|_{maxR}$	D
Diagonal scaling	$A \leftarrow DA,\ A \leftarrow AD,\ A \leftarrow D_L A D_R$	D
	$A \leftarrow DAD$	D
	$A \leftarrow A + BD$	D
Matrix acc and scale	$C \leftarrow \alpha A + \beta B$	D
Matrix add and scale	$B \leftarrow \alpha A + \beta B,\ B \leftarrow \alpha A^T + \beta B$	D

Table 8: Matrix Operations – $O(n^2)$ floating-point operations

Matrix-matrix product	$C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^T B + \beta C$	D,E
	$C \leftarrow \alpha AB^T + \beta C,\ C \leftarrow \alpha A^T B^T + \beta C$	D,E
	$C \leftarrow \alpha AB + C,\ C \leftarrow \alpha A^T B + C$	S
Triangular multiply	$B \leftarrow \alpha TB,\ B \leftarrow \alpha BT$	D,E
	$B \leftarrow \alpha T^T B,\ B \leftarrow \alpha BT^T$	D,E
Triangular solve	$B \leftarrow \alpha T^{-1} B,\ B \leftarrow \alpha T^{-T} B$	D,S,E
	$B \leftarrow \alpha BT^{-1},\ B \leftarrow \alpha BT^{-T}$	D,E
Symmetric rank k & $2k$ updates ($C = C^T$)	$C \leftarrow \alpha AA^T + \beta C,\ C \leftarrow \alpha A^T A + \beta C$	D,E
	$C \leftarrow \alpha AJA^T + \beta C,\ C \leftarrow \alpha A^T JA + \beta C$	D
	$C \leftarrow (\alpha A)B^T + B(\alpha A)^T + \beta C,$ $C \leftarrow (\alpha A)^T B + B^T(\alpha A) + \beta C$	D,E
	$C \leftarrow (\alpha AJ)B^T + B(\alpha AJ)^T + \beta C,$ $C \leftarrow (\alpha AJ)^T B + B^T(\alpha AJ) + \beta C$	D

Table 9: Matrix-Matrix Operations - $O(n^3)$ floating-point operations

Matrix copy	$B \leftarrow A,\ B \leftarrow A^T$	D
Matrix transpose	$A \leftarrow A^T$	D
Permute Matrix	$A \leftarrow PA,\ A \leftarrow AP$	D

Table 10: Data Movement with Matrices

5 Interface Issues

For brevity, the details of language-specific interface issues are not discussed in this paper. Complete details of the design of the language bindings for Fortran 95, Fortran 77, and C, can be found in the respective chapters of the BLAS Technical Forum standard [1]
(`http://www.netlib.org/blas/blast-forum/`).

5.1 Naming Conventions

Language bindings are specified for Fortran 95, Fortran 77, and C.

The Fortran 95 language bindings have routine names of the form <**name**>, where <**name**> is in lowercase letters and indicates the computation performed. These bindings use generic interfaces to manipulate the data type of the routine, and thus their names do not contain a letter to denote the data type.

The Fortran 77 and C language bindings have routine names of the form **BLAS_x**<**name**>, where the letter **x**, indicates the data type as follows:

Data type	x	Fortran 77	x	C
s.p. real	S	REAL	s	float
d.p. real	D	DOUBLE PRECISION	d	double
s.p. complex	C	COMPLEX	c	float
d.p. complex	Z	COMPLEX*16 or DOUBLE COMPLEX	z	double

The suffix <**name**> in the routine name indicates the computation performed. In the matrix-vector and matrix-matrix routines of the Dense and Banded BLAS and the Mixed and Extended-Precision BLAS, the type of the matrix (or of the most significant matrix) is also specified as part of this <**name**> name of the routine. Most of these matrix types apply to both real and complex matrices; a few apply specifically to one or the other, as indicated below.

GB	general band
GE	general (i.e., unsymmetric, in some cases rectangular)
HB	(complex) Hermitian band
HE	(complex) Hermitian
HP	(complex) Hermitian, packed storage
SB	(real) symmetric band
SP	symmetric, packed storage
SY	symmetric
TB	triangular band
TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)

For Fortran 77, routine names are in uppercase letters; however, for the C interfaces all routine names are in lowercase letters. To avoid possible name collisions, programmers are strongly advised not to declare variables or functions with names beginning with these prefixes.

A detailed discussion of the format of the <**name**> naming convention is contained in each respective chapter of the document.

6 Numerical Accuracy and Environmental Enquiry

With a few exceptions, no particular computational order is mandated by the function specifications. In other words, any algorithm that produces results "close enough" to the usual algorithms presented in a standard book on matrix computations [4, 13, 15] is acceptable. For example, Strassen's algorithm may be used for matrix multiplication, even though it can be significantly less accurate than conventional matrix multiplication for some pairs of matrices [15]. Also, matrix multiplication may be implemented either as $C = (\alpha \cdot A) \cdot B + (\beta \cdot C)$ or $C = \alpha \cdot (A \cdot B) + (\beta \cdot C)$ or $C = A \cdot (\alpha \cdot B) + (\beta \cdot C)$, whichever is convenient.

To use the error bounds in [4, 13, 15] and elsewhere, certain machine parameters are needed to describe the accuracy of the arithmetic. Detailed error bounds and limitations due to overflow and underflow are discussed in [5] but all of them depend on details of how floating-point numbers are represented. These details are available by calling an environmental enquiry function called FPINFO. For further details of FPINFO, please refer to the BLAS Technical Forum standard [1]
(`http://www.netlib.org/blas/blast-forum/`).

In the chapters of the BLAS Technical Forum standard, there are exceptional routines where we ask for particularly careful implementations to avoid unnecessary over/underflows, that could make the output unnecessarily inaccurate or unreliable. The details of each routine are described with the language dependent calling sequences. Model implementations that avoid unnecessary over/underflows are based on corresponding LAPACK auxiliary routines, NAG routines, or cited reports.

Floating-point numbers are represented in scientific notation as follows. This discussion follows the IEEE Floating-Point Arithmetic Standard 754 [3].[3]

$$x = \pm d.d \cdots d * BASE^{E}$$

where $d.d \cdots d$ is a number represented as a string of T significant digits in base BASE with the "point" to the right of the leftmost digit, and E is an integer exponent. E ranges from EMIN up to EMAX. This means that the largest representable number, which is also called the *overflow threshold* or OV, is just less than $BASE^{EMAX+1}$, This also means that the smallest positive "normalized" representable number (i.e. where the leading digit of $d.d \cdots d$ is nonzero) is $BASE^{EMIN}$, which is also called the *underflow threshold* or UN.

When overflow occurs (because a computed quantity exceeds OV in absolute value), the result is typically $\pm\infty$, or perhaps an error message. When underflow occurs (because a computed quantity is less than UN in absolute magnitude) the returned result may be either 0 or a tiny number less than UN in magnitude, with minimal exponent EMIN but with a leading zero $(0.d \cdots d)$. Such tiny numbers are often called *denormalized* or *subnormal*, and floating-point arithmetic which returns them instead of 0 is said to support *gradual underflow*.

The *relative machine precision* (or *machine epsilon*) of a basic operation $\odot \in \{+, -, *, /\}$ is defined as the smallest $EPS > 0$ satisfying

$$fl(a \odot b) = (a \odot b) * (1 + \delta) \text{ for some } |\delta| \leq EPS$$

for all arguments a and b that do not cause underflow, overflow, division by zero, or an invalid operation. When $fl(a \odot b)$ is a closest floating-point number to the true result $a \odot b$ (with ties broken arbitrarily), then rounding is called "proper" and $EPS = .5 * BASE^{1-T}$. Otherwise typically $EPS = BASE^{1-T}$, although it can sometimes be worse if arithmetic is not implemented carefully. We further say that rounding is "IEEE style" if ties are broken by rounding to the nearest number whose least significant digit is even (i.e. whose bottom bit is 0).

[3] We ignore implementation details like "hidden bits", as well as unusual representations like logarithmic arithmetic and double-double.

The function FPINFO returns the above floating-point parameters, among others, to help the user understand the accuracy to which results are computed. FPINFO can return the values for either single precision or double precision. The way the precision is specified is language dependent, as is the choice of floating-point parameter to return, and described in Section 6. The names single and double may have different meanings on different machines. We have long been accustomed to single precision meaning 32-bits on all IEEE and most other machines [3], except for Cray and its emulators where single is 64-bits. And there are historical examples of 60-bit formats on some old CDC machines, etc. Nonetheless, we all agree on single precision as a phrase with a certain system-dependent meaning, and double precision too, meaning at least twice as many significant digits as single.

7 Error Handling

The BLAS Technical Forum standard supports two types of error-handling capabilities: an error handler, BLAS_ERROR, and error return codes. Each chapter of the document, and thus each flavor of BLAS, has the choice of using either capability, whichever is more appropriate. The dense and the extended precision BLAS rely on an error handler, and the Sparse BLAS provides error return codes. Each function in the document determines when and if an error-handling mechanism is called, and its function specification must document the conditions (if any) which trigger the error handling mechanism.

For complete details on the error-handling mechanisms available, refer to the BLAS Technical Forum standard [1] (`http://www.netlib.org/blas/blast-forum/`).

8 Acknowledgements

During the period of development of the BLAS Technical Forum Standard, many people served in positions of responsibility and are listed below.

- Jack Dongarra and Sven Hammarling, Conveners and Meeting Chairs
- Susan Blackford and Andrew Lumsdaine, Minutes
- Susan Blackford, Editor

The primary chapter authors are the following:

- Susan Blackford, Jack Dongarra, and Sven Hammarling, Chapter 1
- Susan Blackford, Jack Dongarra, and Sven Hammarling, Linda Kaufman, Zohair Maany, Antoine Petitet, Chapter 2
- Iain Duff, Mike Heroux, Roldan Pozo, Karin Remington, Chapter 3
- Jim Demmel, Greg Henry, Velvel Kahan, Xiaoye Li, Chapter 4
- Clint Whaley, C Interface to the Legacy BLAS
- Jack Dongarra, Fred Krogh, Journal of Development – Environmental routines
- Antoine Petitet, Journal of Development – Distributed-Memory Dense BLAS
- Sven Hammarling, Zohair Maany, Journal of Development – Fortran95 Thin BLAS
- George Corliss, Chenyi Hu, Baker Kearfoot, Bill Walster, J. Wolff v. Gudenberg, Journal of Development – Interval BLAS

We would like to thank the individuals from the following organizations who have written the reference implementations: University of California, Berkeley, University of Houston, Downtown, University of Notre Dame, University of Tennessee, HP/Convex, NAG, NIST, and CERFACS.

Specifically, we thank the following students at the University of California, Berkeley, for their work on the reference implementations and proofreading of various versions of the document: Ben Wanzo, Berkat Tung, Weihua Shen, Anil Kapur, Michael Martin, Jimmy Iskandar, Yozo Hida, Teresa Tung, Yulin Li.

We would like to thank the following vendors and ISPs: Cray, Digital/Compaq, HP/Convex, IBM, Intel, NEC, SGI, Tera, NAG, and VNI.

We thank Paul McMahan of the University of Tennessee for preparing the commenting and voting pages on the BLAS Technical Forum webpage.

We would like to thank the members of the global community who have posted comments, suggestions, and proposals to the email reflector and the BLAS Technical Forum webpage.

And lastly, we would like to thank the attendees of the BLAS Technical Forum meetings:

Andy Anda, Ed Anderson, Zhaojun Bai, David Bailey, Satish Balay, Puri Bangalore, Claus Bendtsen, Jesse Bennett, Mike Berry, Jeff Bilmes, Susan Blackford, Phil Bording, Clay Breshears, Sandra Carney, Mimi Celis, Andrew Chapman, Samar Choudhary, Edmond Chow, Almadena Chtchelkanova, Andrew Cleary, Isom Crawford, Michel Daydé, John Dempsey, Theresa Do, Dave Dodson, Jack Dongarra, Craig Douglas, Paul Dressel, Jeremy Du Croz, Iain Duff, Carter Edwards, Salvatore Filippone, Rob Gjertsen, Roger Golliver, Cormac Garvey, Ian Gladwell, Bruce Greer, Bill Gropp, John Gunnels, Fred Gustavson, Sven Hammarling, Richard Hanson, Hidehiko Hasegawa, Satomi Hasegawa, Greg Henry, Mike Heroux, Jeff Horner, Gary Howell, Mary Beth Hribar, Chenyi Hu, Steve Huss-Lederman, Melody Ivory, Naoki Iwata, Bo Kågström, Velvel Kahan, Chandrika Kamath, Linda Kaufman, David Kincaid, Jim Koehler, Vipin Kumar, Rich Lee, Steve Lee, Guangye Li, Jin Li, Sherry Li, Hsin-Ying Lin, John Liu, Andew Lumsdaine, Dave Mackay, Kristin Marshoffe, Kristi Maschhoff, Brian McCandless, Joan McComb, Noel Nachtigal, Jim Nagy, Esmond Ng, Tom Oppe, Antoine Petitet, Roldan Pozo, Avi Purkayastha, Padma Raghavan, Karin Remington, Yousef Saad, Majed Sidani, Jeremy Siek, Tony Skjellum, Barry Smith, Ken Stanley, Pete Stewart, Shane Story, Chuck Swanson, Françoise Tisseur, Anne Trefethen, Anna Tsao, Robert van de Geijn, Phuong Vu, Kevin Wadleigh, David Walker, Bob Ward, Jerzy Waśniewski, Clint Whaley, Yuan-Jye Jason Wu, Chao Yang, and Guodong Zhang.

Bibliography

[1] BLAS Technical Forum Standard. *The International Journal of High Performance Computing Applications*, this volume:??–??, 2001.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, USA, third edition, 1999. (Also available in Japanese, published by Maruzen, Tokyo, translated by Dr Oguni).

[3] ANSI/IEEE, New York. *IEEE Standard for Binary Floating Point Arithmetic*, Std 754-1985 edition, 1985.

[4] J. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[5] Jim Demmel et al. Extended precision blas paper. *ACM TOMS*, this volume:??–??, 2001.

[6] D. S. Dodson. Corrigendum: Remark on "Algorithm 539: Basic Linear Algebra Subroutines for FORTRAN usage". *ACM Trans. Math. Software*, 9:140, 1983. (See also [18] and [7]).

[7] D. S. Dodson and R. G. Grimes. Remark on algorithm 539: Basic Linear Algebra Subprograms for Fortran usage. *ACM Trans. Math. Software*, 8:403–404, 1982. (See also [18] and [6]).

[8] D. S. Dodson, R. G. Grimes, and J. G. Lewis. Sparse extensions to the FORTRAN Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 17:253–272, 1991. (Algorithm 692).

[9] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[10] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 16:1–28, 1990. (Algorithm 679).

[11] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 14:1–32, 399, 1988. (Algorithm 656).

[12] I. S. Duff, M. Marrone, G. Radicati, and C. Vittoli. Level 3 Basic Linear Algebra Subprograms for sparse matrices: A user-level interface. *ACM Trans. Math. Software*, 23:379–401, 1997.

[13] G. Golub and C. van Loan. *Matrix Computations.* Johns-Hopkins, Baltimore, third edition, 1996.

[14] Sven Hammarling et al. Dense blas paper. *ACM TOMS*, this volume:??–??, 2001.

[15] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, PA, 1996.

[16] Mike Heroux Iain Duff, Roldan Pozo et al. Sparse blas paper. *ACM TOMS*, this volume:??–??, 2001.

[17] IEEE. *ANSI/IEEE Standard for Binary Floating Point Arithmetic: Std 754-1985.* IEEE Press, New York, NY, USA, 1985.

[18] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–323, 1979. (Algorithm 539. See also [7] and [6]).