An Updated Set of Basic Linear Algebra Subprograms (BLAS)

L. SUSAN BLACKFORD
Myricom, Inc.
JAMES DEMMEL
University of California, Berkeley
JACK DONGARRA
The University of Tennessee
IAIN DUFF
Rutherford Appleton Laboratory and CERFACS
SVEN HAMMARLING
Numerical Algorithms Group, Ltd.
GREG HENRY
Intel Corporation
MICHAEL HEROUX
Sandia National Laboratories
LINDA KAUFMAN
William Patterson University
ANDREW LUMSDAINE
Indiana University
ANTOINE PETITET
Sun Microsystems
ROLDAN POZO
National Institute of Standards and Technology
KARIN REMINGTON
The Center for Advancement of Genomics
and
R. CLINT WHALEY
Florida State University

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematical Software**]

General Terms: Algorithms, Standardization

Additional Key Words and Phrases: BLAS, linear algebra, standards

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 computationally intense 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 article 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 BLAS Technical Forum Standard [2002], and on the BLAS Technical Forum webpage (`http://www.netlib.org/blas/blast-forum/`).

Authors' addresses: L. S. Blackford, Myricom, Inc. 325 North Santa Anita Ave., Arcadia, CA 91006; email: susan@myri.com; J. Demmel, Computer Science Division, Department of EECS, 737 Soda Hall, University of California, Berkeley, Berkeley, CA 94720; email: demmel@cs.berkeley.edu; J. Dongarra, University of Tennessee, Department of Computer Science, Suite 203, 1122 Volunteer Blvd., Knoxville, TN 37996-3450; email: dongarra@cs.utk.edu; I. Duff, Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, DIDCOT, Oxon, United Kingdom, OX11 0QX; email: i.s.duff@rl.ac.uk; S. Hammarling, The Numerical Algorithms Group, Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, United Kingdom; email: sven@nag.co.uk; G. Henry, Intel Corporation, Bldg. JF1-13, 2111 NE 25th Avenue, Hillsboro, OR 97124; email: greg.henry@intel.com; M. Heroux, Sandia National Laboratories, 18125 Kreigle Lake Road, Avon, MN 56310; email: mheroux@cs.sandia.gov; L. Kaufman, Computer Science Department, William Patterson University, Wayne, NJ 07470-2103; email: kaufmanl@wpunj.edu; A. Lumsdaine, Computer Science Department, Indiana University, 215 Lindley Hall, Bloomington, IN 47405; email: lums@cs.indiana.edu; A. Petitet, Sun Microsystems, 42, Avenue d'lena, 75016 Paris, France; email: antoine.petitet@sun.com; R. Pozo, National Institute of Standards and Technology, 100 Bureau Drive, Stop 8910, Gaithersburg, MD 20899-8910; email: pozo@nist.gov; K. Remington, The Center for the Advancement of Genomics, 1091 Research Blvd., Rockville, MD 20850; R. C. Whaley, Florida State University, Computer Science Department, 153 Love Building, Tallahassee, FL 32306-4530; email: rwhaley@cs.utk.edu.

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] [Lawson et al. 1979] 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 successfully exploited by LINPACK [Dongarra et al. 1979], a software package for the solution of dense and banded linear equation 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 [Dongarra et al. 1988; 1990], 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 more effectively utilize the memory hierarchy of modern computers. In particular, the Level 3 BLAS allowed the construction of software based upon block-partitioned algorithms, typified by the linear algebra software package LAPACK [Anderson et al. 1999]. LAPACK is state-of-the-art software for the solution of dense and banded linear equations, linear least squares, eigenvalue and singular value problems, 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 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 [Kågström et al. 1998a; 1998b; IBM BLAS; HP BLAS; SUN BLAS Intel BLAS; SGI BLAS].

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 [Dodson et al. 1991; Duff et al. 1997].

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.

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

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 of Order*.[2] 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 conduct discussions on the e-mail 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 Duff et al. [2002] and Li et al. [2002]. 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.

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,[3] Fortran 77 and C. Reference implementations of the standard are provided on the BLAS Technical Forum webpage. 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 article 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 7 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 [Anderson et al. 1999] (e.g., 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

[2]We used a relaxed form of *Robert's Rules of Order* [Robert et al. 2002].
[3]The current Fortran standard.

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.

The original efforts to specify sparse Level 2 and 3 BLAS took considerably longer than the corresponding efforts for the dense and banded BLAS, principally because of the need to obtain agreement on the way to represent sparse matrices. The lessons learned from those efforts have been vital background to the specifications given in the document.

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 [ANSI/IEEE 1985], 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 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 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 [Blackford et al. 2002] as well as Duff et al. [2002].

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 does not impact 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.

The following symbols are used: a, b, c, d, r, s, t, alpha, beta and tau.

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.

Four lowercase letters are used to denote a vector: w, x, y, and z. The corresponding strides are respectively denoted incw, incx, incy, and incz.

Advice to implementors. The increment arguments incw, incx, incy and incz may not be zero. (*End of advice to implementors.*)

Example. The mathematical function returning the inner-product r of two real n-length vectors x and y can be defined by:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i.$$

Rationale. The arguments incx and incy do not play a role in the mathematical formulation of the operation. These arguments allow for the specification of subvector operands in various language bindings. Therefore, some of these arguments may not be present in all language-dependent specifications. (*End of rationale.*)

Table I. Vector Norms

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	$\|x\|_{\infty R}$	$\max_i (\|Re(x_i)\| + \|Im(x_i)\|)$

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 (Table I).

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 $[0 \cdots m-1] \times [0 \cdots n-1]$.

Capital letters are used to denote a matrix. The functions involving matrices use only four symbols, namely A, B, C, and T.

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 (see Table II).

Table II. Matrix Norms

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

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.[4] 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 [Lawson et al. 1979; Dodson et al. 1991; Dongarra et al. 1988; 1990] is a subset of the functionality proposed in this article.

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 article are categorized in a similar manner, with additional categories to cover operations that 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.

[4]In general, the 2-norm of a matrix requires the largest singular value to be computed. A user should see LAPACK, not the BLAS, for this operation.

Table III. Reduction Operations

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 IV. Generate Transformations

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**

The last column of each table denotes in which chapter of the BLAS 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 III lists the scalar and vector reduction operations, Table IV lists the vector rotation operations, Table V lists the vector operations, and Table VI 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.

4.2 Matrix-Vector Operations

This section lists matrix-vector operations in Table VII. 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 third column in the following tables designates G for general, S for symmetirc (Hermitian), or T for triangular matrices. 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.

Table V. Vector Operations

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	$(\ x\ \ y\ ) \leftarrow (\ x\ \ y\ )R$	**D**

Table VI. Data Movement with Vectors

Copy	$y \leftarrow x$	**D**
Swap	$y \leftrightarrow x$	**D**
Sort vector	$x \leftarrow \text{sort}(x)$	**D**
Sort vector & return index vector	$(p, x) \leftarrow \text{sort}(x)$	**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 VII. Matrix–Vector Operations

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

4.3 Matrix Operations

This section lists a variety of matrix operations. The functionality tables are organized as follows. Table VIII lists single matrix operations and matrix operations that involve $O(n^2)$ operations, Table IX lists the $O(n^3)$ matrix–matrix operations and Table X 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 nonunit 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.

Table VIII. Matrix Operations—$O(n^2)$ floating point operations

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

Table IX. Matrix–Matrix Operations—$O(n^3)$ floating point operations

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

Table X. Data Movement with Matrices

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

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 [Blackford 2002].

5.1 Naming Conventions

Routine names have the prefix **BLAS_** to denote the Fortran 77 and C language bindings. The routines in the Fortran 95 language bindings do not contain a prefix. For Fortran 77, all characters are uppercase; however, for the Fortran 95 and C interfaces all characters are lowercase. To avoid possible name collisions, programs are strongly advised not to declare variables or functions with names beginning with these defined prefixes.

The Fortran 77 and C language bindings have names of the form **BLAS_xZZZ**, 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 Fortran 95 language bindings have names of the form **ZZZ**. 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 last letters **ZZZ** indicate the computation performed. In the matrix–vector and matrix–matrix routines the type of the matrix (or of the most significant matrix) is specified in this **ZZZ** name of the routine. Most of these two-letter codes 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)

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

As an example of the language bindings for Fortran 95, Fortran 77, and C we illustrate the interface for matrix multiply. A more complete discussion of the difference and advantages can be found in the BLAS Technical Forum Standard [Blackford et al. 2002].

—Fortran 95 binding:

```
SUBROUTINE gemm( a, b, c [, transa] [, transb] [, alpha] &
                 [, beta] )
  <type>(<wp>), INTENT(IN) :: <aa>, <bb>
  <type>(<wp>), INTENT(INOUT) :: <cc>
  TYPE (blas_trans_type), INTENT(IN), OPTIONAL :: transa,
  transb
  <type>(<wp>), INTENT(IN), OPTIONAL :: alpha, beta
where
  <aa>  ::= a(:,:) or a(:)
  <bb>  ::= b(:,:) or b(:)
  <cc>  ::= c(:,:) or c(:)
and
```

```
c, rank 2, has shape (m,n)
     a has shape (m,k) if transa = blas_no_trans (default)
                 (k,m) if transa /= blas_no_trans
                 (m) if rank 1
     b has shape (k,n) if transb = blas_no_trans (default)
                 (n,k) if transb /= blas_no_trans
                 (n) if rank 1
c, rank 1, has shape (m)
     a has shape (m,n) if transa = blas_no_trans (default)
                 (n,m) if transa /= blas_no_trans
     b has shape (n)
```

—Fortran 77 binding:

```
      SUBROUTINE BLAS_xGEMM( TRANSA, TRANSB, M, N, K, ALPHA, A,
     $                       LDA, B, LDB, BETA, C, LDC )
      INTEGER            K, LDA, LDB, LDC, M, N, TRANSA, TRANSB
      <type>             ALPHA, BETA
      <type>             A( LDA, * ), B( LDB, * ), C( LDC, * )
```

—C binding:

```
void BLAS_xgemm( enum blas_order_type order,
                 enum blas_trans_type transa,
                 enum blas_trans_type transb, int m, int n, int k,
                 SCALAR_IN alpha, const ARRAY a, int lda,
                 const ARRAY b, int ldb, SCALAR_IN beta, ARRAY c,
                 int ldc );
```

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 [Golub and Van Loan 1996; Higham 1996; Demmel 1997] 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 [Higham 1996]. 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 Golub and Van Loan [1996], Demmel [1997], and Higham [1996] 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 Li et al. [2002] 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 [Blackford et al. 2002].

In Section 1.8 of the BLAS Technical Forum standard, there are exception 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 [ANSI/IEEE 1985].[5]

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

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 that 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).

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 is 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

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

all IEEE and most other machines [ANSI/IEEE 1985], 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 standard, 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 standard 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 [Blackford et al. 2002].

ACKNOWLEDGMENTS

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, Introduction

—Susan Blackford, Jack Dongarra, and Sven Hammarling, Linda Kaufman, Zohair Maany, Antoine Petitet, Chapter 2, Dense and Banded BLAS

—Iain Duff, Mike Heroux, Roldan Pozo, Karin Remington, Chapter 3, Sparse BLAS

—Jim Demmel, Greg Henry, Velvel Kahan, Xiaoye Li, Chapter 4, Extended and Mixed Precision BLAS

—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, The 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 standard: 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 e-mail 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.

REFERENCES

HP BLAS. http://www.compaq.com/math/documentation/cxml/dxml.3dxml.html.

IBM BLAS. http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/essl.html.

Intel BLAS. http://developer.intel.com/software/products/mkl/mkl52/index.htm.

SGI BLAS. http://www.sgi.com/software/scsl.html.

SUN BLAS. http://docs.sun.com/htmlcoll/coll.118.3/iso-8859-1/PERFLIBUG/plug_bookTOC.html.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. 1999. *LAPACK Users' Guide*. SIAM,

Philadelphia, PA, USA, third edition. (Also available in Japanese, published by Maruzen, Tokyo, translated by Dr Oguni).

ANSI/IEEE Std 754–1985. *IEEE Standard for Binary Floating Point Arithmetic*.

BLACKFORD, S., CORLISS, G., DEMMEL., J., DONGARRA, J., DUFF, I., HAMMARLING, S., HENRY, G., HEROUX, M., HU, C., KAHAN, W., KAUFMANN, L., KEARFOTT, B., KROGH, F., LI, X., MAANY, Z., PETITET, A., POZO, R., REMINGTON, K., WALSTER, W., WHALEY, C., WOLFF, V., GUDENBERG, J.,AND LUMSDAINE, A. 2002. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. *Int. J. High Perform. Comput. 16*, 1–2. (also available at www.netlib.org/blas/blast-forum).

DEMMEL, J. 1997. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, Pa.

DODSON, D. S. 1983. Corrigendum: Remark on "Algorithm 539: Basic linear algebra subroutines for FORTRAN usage". *ACM Trans. Math. Software 9*, 140. (See also Lawson et al. [1979] and Dodson and Grimes [1982]).

DODSON, D. S. AND GRIMES, R. G. 1982. Remark on algorithm 539: Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw. 8*, 403–404. (See also Lawson et al. [1979] and Dodson [1983]).

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*, 253–272. (Algorithm 692).

DONGARRA, J. J., BUNCH, J. R., MOLER, C. B., AND STEWART, G. W. 1979. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, Pa.

DONGARRA, J. J., DU CROZ, J., DUFF, I. S., AND HAMMARLING, S. 1990. A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1–28. (Algorithm 679).

DONGARRA, J. J., DU CROZ, J., HAMMARLING, S., AND HANSON, R. J. 1988. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw. 14*, 1–32, 399. (Algorithm 656).

DUFF, I. S., HEROUX, M. A., AND POZO, R. 2002. An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS Technical Forum. *ACM Trans. Math. Softw. 28*, 2 (June), 000–000.

DUFF, I. S., MARRONE, M., RADICATI, G., AND VITTOLI, C. 1997. Level 3 basic linear algebra subprograms for sparse matrics: A user-level interface. *ACM Trans. Math. Softw. 23*, 379–401.

GOLUB, G. AND VAN LOAN, C. 1996. *Matrix Computations*. 3rd ed. Johns-Hopkins, Baltimore, Md.

HIGHAM, N. J. 1996. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, Pa.

KÅGSTRÖM, B., LING, P., AND VAN LOAN, C. 1998a GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Softw. 24*, 3, 268–302.

KÅGSTRÖM, B., LING, P., AND VAN LOAN, C. 1998b. Algorithm 784: GEMM-based level 3 BLAS: Portability and optimization issues. *ACM Trans. Math. Softw. 24*, 3, 303–316.

LAWSON, C. L., HANSON, R. J., KINCAID, D., AND KROGH, F. T. 1979. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Softw. 5*, 308–323. (Algorithm 539. See also Dodson and Grimes [1982] and Dodson [1983].).

LI X. S., DEMMEL, J. W., BAILEY, D. H., HENRY, G., HIDA, Y., ISKANDAR, J., KAHAN, W., KANG, S. Y., KAPUR, A., MARTIN, M. C., THOMPSON, B. J., TUNG, T., AND YOD, D. J. 2002. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Softw. 28*, 2 (June), 000–000.

ROBERT III, H. M., EVANS, W. J., HONEMANN, D. H., AND BALCH, T. J. 2000. *Robert's Rules of Order* 10th ed. Perseus Book Group

Received April 2002; revised May 2002; accepted June 2002

Update Cancel. addSJdt WQboyVARL wEziJRWaPnHglXQMiiuTmAnAgMAs owBgExufTXoeldTvla,GIGG LsJkLGLgxCAvqk.Â Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, 100 views Â· View 1 Upvoter. View more. Related Questions. In order to learn machine learning, do I need all the knowledge in linear algebra or should I focus on some specific concepts? I want to look more into linear algebra, what should I review first? What is a basis in linear algebra? How is linear algebra used in AI? What parts of linear algebra are relevant to computer science? Why is linear algebra useful? The rst part of the name, Basic Linear Algebra Subroutines, echoes BLAS, which stands for Basic Linear Algebra Subprograms [15]. The new word â€Subroutinesâ€ indicates a key imple-mentation feature of BLASFEO: the use of a modular design, based on assembly subroutines (as explained in detail in Section 4.4).Â Linear algebra routines are a key aspect in the implementation of these solvers, since they perform the most computationally expensive operations.Â A set of linear algebra routines tailored to embedded optimization problems can take advantage of the special features of this class of problems in order to reduce the computational time.Â The wrapper simply takes care of extracting this information and updating the pointer to the rst element of the working sub-matrix, as. Additional Key Words and Phrases: BLAS, linear algebra, standards. 1. INTRODUCTION. Numerical linear algebra, particularly the solution of linear systems of equa-. tions, linear least squares problems, eigenvalue problems and singular value. ACM Transactions on Mathematical Software, Vol. 28, No. 2, June 2002, Pages 135â€“151.Â of a set of kernel routines for linear algebra, historically called the Basic Linear. Algebra Subprograms and commonly known as the BLAS. The complete stan-. dard can be found in BLAS Technical Forum Standard [2002], and on the BLAS.