Basic Linear Algebra Subprograms for Fortran Usage

C. L. LAWSON
Jet Propulsion Laboratory
R. J. HANSON
Sandia Laboratories
D. R. KINCAID
The University of Texas at Austin
and
F. T. KROGH
Jet Propulsion Laboratory

A package of 38 low level subprograms for many of the basic operations of numerical linear algebra is presented. The package is intended to be used with Fortran. The operations in the package include dot product, elementary vector operation, Givens transformation, vector copy and swap, vector norm, vector scaling, and the determination of the index of the vector component of largest magnitude.

The subprograms and a test driver are available in portable Fortran. Versions of the subprograms are also provided in assembly language for the IBM 360/67, the CDC 6600 and CDC 7600, and the Univac 1108.

Key Words and Phrases: linear algebra, utilities
CR Categories: 4.49, 5.14
The Algorithm: Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Software 5*, 3 (Sept. 1979), 324–325.

1. INTRODUCTION

This paper describes a package, called the BLAS (Basic Linear Algebra Subprograms), of 38 Fortran-callable subprograms for basic operations of numerical linear algebra. This paper and the associated package of subprograms and testing

The work of the first and fourth authors was supported by the National Aeronautics and Space Administration under Contract NAS 7-100. The work of the second author was supported by the U.S. Energy Research and Development Administration (ERDA) under Contract AT(29-1)-789 and (at Washington State University) by the Office of Naval Research under Contract NR 044-457.

Authors' addresses: C.L. Lawson, Jet Propulsion Laboratory, M/S 125-128, 4800 Oak Grove Drive, Pasadena, CA 91103; R.J. Hanson, Numerical Mathematics, Div. 5122, Sandia Laboratories, Albuquerque, NM 87115; D.R. Kincaid, Center for Numerical Analysis, The University of Texas at Austin, Austin, TX 78712; F.T. Krogh, Jet Propulsion Laboratory, M/S 125-128, 4800 Oak Grove Drive, Pasadena, CA 91103.

programs are the result of a collaborative voluntary project of the ACM-SIGNUM committee on basic linear algebra subprograms. This project was carried out during the period 1973–1977.

The initial version of the subprogram specifications appeared in [11]. Following distribution of [11] to persons active in the development of numerical linear algebra software, open meetings of the project were held at the Mathematical Software II Conference, Purdue University, May 1974 [13], and at the National Computer Conference, Anaheim, May 1975. Extensive modifications of the specifications were made following the Purdue meeting, which was attended by 30 people. A few additional changes resulted from the Anaheim meeting. Most of the further Fortran code changes resulted from an effort to improve the design and to make the subroutines more robust.

2. REASONS FOR DEVELOPING THE PACKAGE

Designers of computer programs involving linear algebraic operations have frequently chosen to implement certain low level operations such as the dot product as separate subprograms. This may be observed both in many published codes and in codes written for specific applications at many computer installations. Following are some of the reasons for taking this approach:

(1) It can serve as a conceptual aid in both the design and coding stages of a programming effort to regard an operation such as the dot product as a basic building block. This is consistent with the ideas of structured programming which encourage modularizing common code sequences.

(2) It improves the self-documenting quality of code to identify an operation such as the dot product by a unique mnemonic name.

(3) Since a significant amount of the execution time in complicated linear algebraic programs may be spent in a few low level operations, a reduction of the execution time spent in these operations may be reflected in cost savings in the running of programs. Assembly language coded subprograms for these operations provide such savings on some computers.

(4) The programming of some of these low level operations involves algorithmic and implementation subtleties that are likely to be ignored in the typical applications programming environment. For example, the subprograms provided for the modified Givens transformation incorporate control of the scaling terms, which otherwise can drift monotonically toward underflow.

If there could be general agreement on standard names and parameter lists for some of these basic operations, it would add the additional benefit of *portability* with *efficiency* on the assumption that the assembly language subprograms were generally available. Such standard subprograms would provide building blocks with which designers of portable subprograms for higher level linear algebraic operations such as solving linear algebraic equations, eigenvalue problems, etc., could achieve additional efficiency. The package of subprograms described in this paper is proposed to serve this purpose.

3. SCOPE OF THE PACKAGE

Specifications will be given for 38 Fortran-callable subprograms covering the operations of dot product, vector plus a scalar times a vector, Givens transfor-

mation, modified Givens transformation, copy, swap, Euclidean norm, sum of magnitudes, multiplying a scalar times a vector, and locating an element of largest magnitude. Since we are thinking of these subprograms as being used in an ANSI FORTRAN context, we provide for the cases of single precision, double precision, and (single precision) complex data.

In Table I a concise summary of the operations provided and the conventions adopted for naming the subprograms is given. Each type of operation is identified by a root name. The root name is prefixed by one or more of the letters I, S, D, C, or Q to denote operations on integer, single precision, double precision, (single precision) complex, or extended precision data types, respectively. For subprograms involving a mixture of data types the type of the output quantity is indicated by the leftmost prefix letter. Suffix letters are used on four of the dot product subprograms to distinguish variants of the basic operation.

If one were to extend this package to include double precision complex type data (COMPLEX * 16 in IBM Fortran), we suggest that the prefix Z be used in the names of the new subprograms. For example, subprograms CZDOTC and CZDOTU for the dot product of (single precision) complex vectors, with double precision accumulation, have been written for the CDC 6600. These may be obtained directly from Kincaid.

Section 5 lists all of the subprogram names and their parameter lists, and defines the operations performed by each subprogram.

The criterion for including an operation in the package was that it should involve just one level of looping and occur in the usual algorithms of numerical linear algebra, such as Gaussian elimination or the various elimination methods using orthogonal transformations.

Table I. Summary of Functions and Names of the BLAS Subprograms

Function	Prefix and suffix of name								Root of name
Dot product	SDS-	DS-	DQ-I	DQ-A	C-U	C-C	D-	S-	-DOT-
Constant times a vector plus a vector						C-	D-	S-	-AXPY
Set up Givens rotation							D-	S-	-ROTG
Apply rotation							D-	S-	-ROT
Set up modified Givens rotation							D-	S-	-ROTMG
Apply modified rotation							D-	S-	-ROTM
Copy x into y						C-	D-	S-	-COPY
Swap x and y						C-	D-	S-	-SWAP
2-norm (Euclidean length)						SC-	D-	S-	-NRM2
Sum of absolute values[a]						SC-	D-	S-	-ASUM
Constant times a vector					CS-	C-	D-	S-	-SCAL
Index of element having maximum absolute value[a]						IC-	ID-	IS-	-AMAX

[a] For complex components $z_j = x_j + iy_j$ these subprograms compute $|x_j| + |y_j|$ instead of $(x_j^2 + y_j^2)^{1/2}$.

This orientation affected the specifications of SCASUM and ICAMAX particularly. Although SASUM and DASUM compute ℓ_1 norms, we assumed that the usage of either of these subprograms in numerical linear algebra software would be for the purpose of computing a vector norm that was less expensive to compute than the ℓ_2 norm. Thus for the complex version, SCASUM, instead of specifying the ℓ_1 norm which would be

$$w = \sum_i \{[\mathrm{Re}(x_i)]^2 + [\mathrm{Im}(x_i)]^2\}^{1/2},$$

we specified the less expensive function,

$$w = \sum_i \{|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)|\}.$$

Similarly, whereas ISAMAX and IDAMAX may be regarded as determining the ℓ_∞ norm of a vector, we do not regard this as the essential property to be carried over to the complex case. Thus ICAMAX is specified to find an index j such that

$$|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| = \max_i \{|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)|$$

rather than finding an index j such that

$$[\mathrm{Re}(x_j)]^2 + [\mathrm{Im}(x_j)]^2 = \max_i \{[\mathrm{Re}(x_i)]^2 + [\mathrm{Im}(x_i)]^2.$$

Let $\|x\|_{\tilde{1}}$ denote the function computed by SCASUM, and define $\|x\|_{\tilde{\infty}} = |\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)|$ where j is the index found by ICAMAX. The size functions $\|x\|_{\tilde{1}}$ and $\|x\|_{\tilde{\infty}}$ in are approximations to the classical norms $\|x\|_1$ and $\|x\|_\infty$ in the sense that $\|x\|_1 \le \|x\|_{\tilde{1}} \le 2^{1/2}\|x\|_1$ and $\|x\|_\infty \le \|x\|_{\tilde{\infty}} \le 2^{1/2}\|x\|_\infty$.

Furthermore, if one defines related size functions for matrices by

$$\|A\|_{\tilde{1}} = \max_j \|a_{\cdot j}\|_{\tilde{1}}$$

and

$$\|A\|_{\tilde{\infty}} = \max_i \|a_{i\cdot}\|_{\tilde{1}},$$

then the following consistency relations can be verified:

$$\|A\|_{\tilde{1}} = \max\{\|Ax\|_{\tilde{1}} : \|x\|_{\tilde{1}} = 1\}$$

and

$$\|A\|_{\tilde{\infty}} = \max\{\|Ax\|_{\tilde{\infty}} : \|x\|_{\tilde{\infty}} = 1\}.$$

Although these size functions satisfy $\|\alpha x\| = |\alpha| \cdot \|x\|$ for real α, they do not satisfy such homogeneity conditions for arbitrary complex α. They do, however, satisfy the bounds $2^{-1/2}|\alpha| \cdot \|x\|_{\tilde{1}} \le \|\alpha x\|_{\tilde{1}} \le 2^{1/2}|\alpha| \cdot \|x\|_{\tilde{1}}$ and $2^{-1/2}|\alpha| \cdot \|x\|_{\tilde{\infty}} \le \|\alpha x\|_{\tilde{\infty}} \le 2^{1/2}|\alpha| \cdot \|x\|_{\tilde{\infty}}$ for complex α.

In both the computation of the ℓ_2 norm and the Givens transformation a naïve computation of the squares of the given data would restrict the exponent range of acceptable data. This package avoids this restriction by making use of ideas described by Cody [7] and Blue [5]. Additionally, in the case of the Givens transformation, an idea of Stewart [15] permits the storage of all the transformations of a matrix decomposition in the memory space occupied by the elements zeroed by the transformation.

The modified Givens transformation is a relatively new innovation among numerical linear algebra algorithms [9, 10, 14]. The significant features are the reduction of the number of multiplications, the elimination of square root operations, and the capability of removing rows of data in least squares problems. The details of this algorithm as implemented in this package are given in the Appendix.

4. PROGRAMMING CONVENTIONS

Vector arguments are permitted to have a storage spacing between elements. This spacing is specified by an increment parameter. For example, suppose a vector x having components x_i, $i = 1, \ldots, N$, is stored in a DOUBLE PRECISION array DX( ) with increment parameter INCX. If INCX ≥ 0 then x_i is stored in DX(1 + (i − 1)*INCX). If INCX < 0 then x_i is stored in DX(1 + (N − i)*|INCX|). This method of indexing when INCX < 0 avoids negative indices in the array DX( ) and thus permits the subprograms to be written in Fortran. Only positive values of INCX are allowed for operations 26–38 in Section 5 that each have a single vector argument.

It is intended that the loops in all subprograms process the elements of vector arguments in order of increasing vector component indices, i.e. in the order x_i, $i = 1, \ldots, N$. This implies processing in reverse storage order when INCX < 0. If these subprograms are implemented on a computer having parallel processing capability, it is recommended that this order of processing be adhered to as nearly as is reasonable.

5. SPECIFICATION OF THE BLAS SUBPROGRAMS

Type and dimension information for variables occurring in the subprogram specifications are as follows:

$$mx = \max(1, N*|INCX|), \qquad my = \max(1, N*|INCY|).$$

```
INTEGER   N, INCX, INCY, IMAX, QC(10)
REAL   SX(mx), SY(my), SA, SB, SC, SS
REAL   SD1, SD2, SB1, SB2, SPARAM(5), SW
DOUBLE PRECISION   DX(mx), DY(my), DA, DB, DC, DS
DOUBLE PRECISION   DD1, DD2, DB1, DB2, DPARAM(5), DW
COMPLEX   CX(mx), CY(my), CA, CW
```

Type declarations for function names are as follows:

```
INTEGER   ISAMAX, IDAMAX, ICAMAX
REAL   SDOT, SDSDOT, SNRM2, SCNRM2, SASUM, SCASUM
DOUBLE PRECISION   DSDOT, DDOT, DQDOTI, DQDOTA, DNRM2, DASUM
COMPLEX   CDOTC CDOTU
```

Dot Product Subprograms

1. SW = SDOT(N, SX, INCX, SY, INCY),

$$w := \sum_{i=1}^{N} x_i y_i.$$

2. DW = DSDOT(N, SX, INCX, SY, INCY),

$$w := \sum_{i=1}^{N} x_i y_i.$$

Double precision accumulation is used within the subprogram DSDOT.

3. SW = SDSDOT(N, SB, SX, INCX, SY, INCY),

$$w := b + \sum_{i=1}^{N} x_i y_i.$$

The accumulation of the inner product and the addition of b are in double precision. The conversion of the final result to single precision is done in the same way as the intrinsic function SNGL( ).

4. DW = DDOT(N, DX, INCX, DY, INCY),

$$w := \sum_{i=1}^{N} x_i y_i.$$

5. DW = DQDOTI(N, DB, QC, DX, INCX, DY, INCY),

$$w := b + \sum_{i=1}^{N} x_i y_i.$$

The input data, b, x, and y, are converted internally to extended precision. The result is stored in extended precision form in QC( ) and returned in double precision form as the value of the function DQDOTI.

6. DW = DQDOTA(N, DB, QC, DX, INCX, DY, INCY),

$$w := c := b + c + \sum_{i=1}^{N} x_i y_i.$$

The input value of c in QC( ) is in extended precision. The value c must have resulted from a previous execution of DQDOTI or DQDOTA since no other way is provided for defining an extended precision number. The computation is done in extended precision arithmetic and the result is stored in extended precision form in QC( ) and is returned in double precision form as the function value DQDOTA.

7. CW = CDOTC(N, CX, INCX, CY, INCY),

$$w := \sum_{i=1}^{N} \bar{x}_i y_i.$$

The suffix C on CDOTC indicates that the complex conjugates of the components x_i are used.

8. CW = CDOTU(N, CX, INCX, CY, INCY),

$$w := \sum_{i=1}^{N} x_i y_i.$$

The suffix U on CDOTU indicates that the vector components x_i are used unconjugated.

In the preceding eight subprograms the value of $\sum_{i=1}^{N}$ will be set to zero if $N \leq 0$.

Elementary Vector Operation: $y := ax + y$

9. CALL SAXPY(N, SA, SX, INCX, SY, INCY).
10. CALL DAXPY(N, DA, DX, INCX, DY, INCY).

11. CALL CAXPY(N, CA, CX, INCX, CY, INCY).

If $a = 0$ or if $N \leq 0$ these subroutines return immediately.

Construct Givens Plane Rotation

12. CALL SROTG(SA, SB, SC, SS).
13. CALL DROTG(DA, DB, DC, DS).

Given a and b, each of these subroutines computes

$$\sigma = \begin{cases} \operatorname{sgn}(a) & \text{if } |a| > |b|, \\ \operatorname{sgn}(b) & \text{if } |b| \geq |a|, \end{cases} \qquad r = \sigma(a^2 + b^2)^{1/2},$$

$$c = \begin{cases} a/r & \text{if } r \neq 0, \\ 1 & \text{if } r = 0, \end{cases} \qquad s = \begin{cases} b/r & \text{if } r \neq 0, \\ 0 & \text{if } r = 0. \end{cases}$$

The numbers c, s, and r then satisfy the matrix equation

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}.$$

The introduction of σ is not essential to the computation of a Givens rotation matrix, but its use permits later stable reconstruction of c and s from just one stored number, an idea due to Stewart [15]. For this purpose the subroutine also computes

$$z = \begin{cases} s & \text{if } |a| > |b|, \\ 1/c & \text{if } |b| \geq |a| \quad \text{and} \quad c \neq 0, \\ 1 & \text{if } c = 0. \end{cases}$$

The subroutines return r overwriting a, and z overwriting b, as well as returning c and s.

If the user later wishes to reconstruct c and s from z, it can be done as follows:

If $z = 1$ set $c = 0$ and $s = 1$.
If $|z| < 1$ set $c = (1 - z^2)^{1/2}$ and $s = z$.
If $|z| > 1$ set $c = 1/z$ and $s = (1 - c^2)^{1/2}$.

Apply a Plane Rotation

14. CALL SROT(N, SX, INCX, SY, INCY, SC, SS).
15. CALL DROT(N, DX, INCX, DY, INCY, DC, DS).

Each of these subroutines computes

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} := \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{for} \quad i = 1, \ldots, N.$$

If $N \leq 0$ or if $c = 1$ and $s = 0$ the subroutines return immediately.

Construct a Modified Givens Transformation

16. CALL SROTMG(SD1, SD2, SB1, SB2, SPARAM).
17. CALL DROTMG(DD1, DD2, DB1, DB2, DPARAM).

The input quantities d_1, d_2, b_1, and b_2 define a 2-vector $[a_1, a_2]^T$ in partitioned form as

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} d_1^{1/2} & 0 \\ 0 & d_2^{1/2} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

The subroutine determines the modified Givens rotation matrix H, as defined in eqs. (A6) and (A7) of the Appendix, that transforms b_2, and thus a_2, to zero. A representation of this matrix is stored in the array SPARAM( ) or DPARAM( ) as follows. Locations in PARAM not listed are left unchanged.

PARAM(1) = 1, case of eq. (A7)	PARAM(1) = 0, case of eq. (A6)	PARAM(1) = −1, case of rescaling
$h_{12} = 1$, $h_{21} = -1$	$h_{11} = h_{22} = 1$	PARAM(2) = h_{11}
PARAM(2) = h_{11}	PARAM(3) = h_{21}	PARAM(3) = h_{21}
PARAM(5) = h_{22}	PARAM(4) = h_{12}	PARAM(4) = h_{12}
		PARAM(5) = h_{22}

In addition PARAM(1) = −2 indicates $H = I$.

The values of d_1, d_2, and b_1 are changed to represent the effect of the transformation. The quantity b_2 which would be zeroed by the transformation is left unchanged in storage.

The input value of d_1 should be nonnegative, but d_2 can be negative for the purpose of removing data from a least squares problem. Further details can be found in the Appendix.

Apply a Modified Givens Transformation

18. CALL SROTM(N, SX, INCX, SY, INCY, SPARAM).
19. CALL DROTM(N, DX, INCX, DY, INCY, DPARAM).

Let H denote the modified Givens transformation defined by the parameter array SPARAM( ) or DPARAM( ). The subroutines compute

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} := H \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{for} \quad i = 1, \ldots, \text{N}.$$

If N ≤ 0 or if H is an identity matrix the subroutines return immediately. See the Appendix for further details.

Copy a Vector x to y: y := x

20. CALL SCOPY(N, SX, INCX, SY, INCY).
21. CALL DCOPY(N, DX, INCX, DY, INCY).
22. CALL CCOPY(N, CX, INCX, CY, INCY).

Return immediately if N ≤ 0.

Interchange Vectors x and y: x :=: y

23. CALL SSWAP(N, SX, INCX, SY, INCY).
24. CALL DSWAP(N, DX, INCX, DY, INCY).
25. CALL CSWAP(N, CX, INCX, CY, INCY).

Return immediately if N ≤ 0.

Euclidean Length or ℓ_2 Norm of a Vector

$$w := \left[\sum_{i=1}^{N} |x_i|^2 \right]^{1/2}$$

26. SW = SNRM2(N, SX, INCX).
27. DW = DNRM2(N, DX, INCX).
28. SW = SCNRM2(N, CX, INCX).

If $N \leq 0$ the result is set to zero.

Sum of Magnitudes of Vector Components

29. SW = SASUM(N, SX, INCX).
30. DW = DASUM(N, DX, INCX).
31. SW = SCASUM(N, CX, INCX).

The functions SASUM and DASUM compute $w := \sum_{i=1}^{N} |x_i|$. The function SCASUM computes

$$w := \sum_{i=1}^{N} \{|\mathrm{Re}(x_i)| + |\mathrm{Im}(x_i)|\}.$$

These functions return immediately with the result set to zero if $N \leq 0$.

Vector Scaling: $x := ax$

32. CALL SSCAL(N, SA, SX, INCX).
33. CALL DSCAL(N, DA, DX, INCX).
34. CALL CSCAL(N, CA, CX, INCX).
35. CALL CSSCAL(N, SA, CX, INCX).

Return immediately if $N \leq 0$.

Find Largest Component of a Vector

36. IMAX = ISAMAX(N, SX, INCX).
37. IMAX = IDAMAX(N, DX, INCX).
38. IMAX = ICAMAX(N, CX, INCX).

The functions ISAMAX and IDAMAX determine the smallest index i such that $|x_i| = \max\{|x_j| : j = 1, \ldots, N\}$.

The function ICAMAX determines the smallest index i such that $|x_i| = \max\{|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| : j = 1, \ldots, N\}$.

These functions set the result to zero and return immediately if $N \leq 0$.

6. IMPLEMENTATION

In addition to the Fortran versions, all of the subprograms except DQDOTI and DQDOTA are also supplied in assembler language for the Univac 1108, the IBM 360/67, and the CDC 6600 and 7600. The Fortran versions of DQDOTI and DQDOTA use part of Brent's multiple precision package [6]. Assembler language modules for these two subprograms are given only for the Univac 1108.

Only four of the assembly routines for the CDC 6600 and 7600 take advantage of the pipeline architecture of these machines. The four routines SDOT( ),

SAXPY( ), SROT( ), and SROTM( ) are those typically used in the innermost loop of computations. Some timing results are given in Section 8.

7. RELATION TO THE ANSI FORTRAN STANDARD

American National Standard Fortran, ANSI X3.9-1966 [1, 3, 4], which will be referred to as 1966 FORTRAN, is widely supported by existing Fortran compilers. We will refer to American National Standard Fortran, ANSI X3.9-1977 [2], as FORTRAN 77.

The calling sequences of the BLAS subprograms would require that the subprograms contain declarations of the form

```
REAL SX(MAX0(1, N*IABS(INCX)))
```

to precisely specify the array lengths. Neither 1966 FORTRAN nor FORTRAN 77 permits such a statement. A statement of the form REAL SX(1) is permitted by major Fortran compilers to cover cases in which it is inconvenient to specify an exact dimension. This latter form is used in the BLAS subprograms even though it does not conform to 1966 FORTRAN. FORTRAN 77 allows the form REAL SX(*) for this situation. Thus the BLAS package can be made to conform to FORTRAN 77 by changing 1's to *'s in the subprogram array declarations.

8. TESTING

A Master Test Program has been written in Fortran and is included with the submitted code. This package consists of a main program and a set of subprograms containing built-in test data and correct answers. It executes a fixed set of test cases exercising all 38 subprograms or optionally any selected subset of these.

We have attempted to design the test cases and the Master Test Program to be usable on a wide variety of nondecimal machines having Fortran systems.

The Master Test Program has been successfully executed, testing the Fortran coded version of the BLAS subprograms, on Univac 1108, IBM 360/67, Burroughs 6700, CDC 6600, and CDC 7600 computers. These tests have also been run successfully testing the respective assembler packages on the Univac 1108, IBM 360/67, CDC 6600, and CDC 7600 computers.

The following method of comparing true and computed numbers is used in the Master Test Program. Let z denote a prestored true result and let $\bar{z}$ denote the corresponding computed result to be tested. The numbers σ and ϕ are prestored constants that will be discussed below. The test program computes

$$d = \mathrm{fl}(\bar{z} - z), \quad g = \mathrm{fl}(|\sigma| + |\mathrm{fl}(\phi * d)|), \quad h = |\sigma|, \quad \tau = \mathrm{fl}(g - h),$$

where fl denotes machine floating-point arithmetic of the current working precision, either single precision or double precision. It is further assumed that g and h are truncated to working precision before being used in the computation of τ.

The test is passed if $\tau = 0$ and fails if $\tau \neq 0$. Note that τ will be zero if $|d|$ is so small that adding $|\mathrm{fl}(\phi * d)|$ to $|\sigma|$ gives a result that is not distinguished from $|\sigma|$ when truncated to working precision.

For example, suppose $\sigma = 1.0$, $\phi = 0.5$, $d = 10^{-9}$; then the mathematical value of $\sigma + \phi * d$ is 1.0000000005, but the single precision computed value of g on the

Univac 1108 will be 1.0, resulting in $\tau = 0$. Thus in this case d is small enough to pass the test.

The number σ is prestored along with the correct result z in the testing program. In general σ has different values for different test cases.

The number ϕ is a "tuning" factor which has been determined empirically to make the test perform correctly on a variety of machines. Note that the stringency of the test is relaxed by decreasing the value of ϕ. This has been used to desensitize the testing to the effects of differences in the treatment of trailing digits in the floating-point arithmetic of different machines.

There are four different values of ϕ prestored in the main program, TBLA, of the testing package. These values are called SFAC, SDFAC, DFAC, and DQFAC. These are used for testing operations which are respectively single precision, mixed single and double precision, double precision, and mixed double and extended precision.

It is intended that the test package be useful to anyone who undertakes the implementation of an assembly coded version of this package. In working on a new machine, one may find it necessary to reduce the values of one or more of the numbers SFAC, SDFAC, DFAC, or DQFAC to obtain correct test performance. The authors would appreciate hearing of any new assembly coded versions of the packages and of any need to reduce the values of these tuning parameters.

9. SELECTED RESULTS FOR THE IBM 360/67, CDC 6600, AND UNIVAC 1108 TIMING OF DOT PRODUCTS AND ELEMENTARY VECTOR OPERATIONS

The most obvious implementation of the dot product and elementary vector operations for vectors with unit storage increments are in-line Fortran loops 1 and 2:

```
      W = 0.
      DO 10 I = 1, N
10    W = W + X(I) * Y(I)
```

In-line Fortran for dot products, loop 1

```
      DO 20 I = 1, N
20    Y(I) = A * X(I) + Y(I)
```

In-line Fortran for elementary vector operations, loop 2

The BLAS replacements for these in-line Fortran loops, using the same variable names and appropriate type statements, are

```
W = __DOT(N, X, 1, Y, 1)
```

BLAS replacement for loop 1

```
CALL __AXPY(N, A, X, 1, Y, 1)
```

BLAS replacement for loop 2

Table II. __DOT( ) Function and In-Line Loop 1 Timings

Time, in seconds, is given for 1000 executions of in-line Fortran loop 1 and calls to the __DOT( ) function. Times for 5 runs were averaged.

Apply factors of 1.1 and 0.75 to IBM 360/67 times to get approximate respective times for nonequally spaced increments and single precision. No distinction for nonequal increments is necessary for the CDC 6600 and Univac 1108.

	IBM 360/67, double precision		CDC 6600, single precision		Univac 1108, single precision	
Vector length, N	In-line Fortran (H, Opt = 2)	Assembler	In-line Fortran (FTN, Opt = 2)	Assembler	In-line Fortran	Assembler
10	0.1438	0.1917	0.0360	0.0480	0.0756	0.0790
25	0.3436	0.3854	0.0750	0.0625	0.1836	0.1730
50	0.6719	0.7186	0.1400	0.0800	0.3598	0.3182
100	1.3750	1.3750	0.2800	0.1250	0.6986	0.6162

Table III. __AXPY( ) Subprogram and In-Line Loop 2 Timings

Time, in seconds, is given for 1000 executions of in-line Fortran loop 2 and calls to the __AXPY( ) subprogram. Times for 5 runs were averaged.

Apply factor of 0.75 to get single precision IBM 360/67 times. Only vectors with unit increments were used in this timing.

	IBM 360/67, double precision		CDC 6600, single precision		Univac 1108, single precision	
Vector length, N	In-Line Fortran (H, Opt = 2)	Assembler	In-line Fortran (FTN, Opt = 2)	Assembler	In-line Fortran	Assembler
10	0.0590	0.2050	0.0500	0.0650	0.0740	0.0886
25	0.3930	0.4375	0.1125	0.1000	0.1806	0.1890
50	0.7950	0.8400	0.2100	0.1725	0.3544	0.3574
100	1.5500	1.6000	0.4200	0.3000	0.7292	0.7170

The " " in front of the BLAS subprogram names is due to the fact that both single and double precision versions are discussed here.

These subprograms, coded in assembly language, were timed and compared with the time for the in-line loops. As was stated in Section 2, one reason for the development of the package was to make highly efficient code possible. This goal has been achieved for the CDC 6600 but not for the IBM 360/67. The IBM 360/67 Fortran H compiler, operating with Opt = 2, generates nearly perfect object code.

In Tables II and III we give some sample times for the three machines; loops 1 and 2 and their BLAS replacements are compared. Interpretation of Tables II and III, supported more fully in [12], are as follows:

—Because of linkage overhead, the BLAS subprograms for the IBM 360/67 are always less efficient than the in-line loops. For vectors of large enough length the linkage overhead is relatively negligible.

—The dot product and elementary vector operation subprograms for the CDC

Table IV. Standard and Modified Givens Transformation in Matrix Triangularization

Time, in seconds, is given for the triangularization of 2N by N matrices using standard and modified Givens transformations. Times for 5 runs were averaged.

	IBM 360/67, double precision		CDC 6600, single precision		Univac 1108, single precision	
N	Standard Givens	Modified Givens	Standard Givens	Modified Givens	Standard Givens	Modified Givens
10	0.0800	0.0650	0.0200	0.0190	0.0335	0.0298
25	0.8789	0.6250	0.1719	0.1445	0.3633	0.3001

6600 are respectively 3.1 and 1.6 times more efficient than in-line code for vectors of large enough length.

—For the CDC 6600, dot products are considerably more efficient than elementary vector operations on vectors of the same length.

Timing of Standard and Modified Givens Methods. Gentleman's modification of the Givens transformation [9] is discussed in the Appendix. This technique eliminates square roots and two of the four multiply operations when forming the product of the resulting matrix by a 2-vector.

The relative efficiency of Gentleman's modification to the standard Givens transformation was compared. Both techniques were used to triangularize 2N by N matrices $A = \{a_{ij}\}$ where

$$a_{ij} = (i + j - 1)^{-1}$$

In Table IV there are some sample times which resulted from the triangularizations using both methods.

We are primarily interested in algorithm comparison here, so both methods were timed using their assembler versions to apply the matrix products.

A conclusion is that in the context of triangularizing matrices, the modified Givens transformation method is ultimately more efficient in computer time by factors varying between 1.4 and 1.6. This is fully supported in [12]. The comparison is most favorable on the IBM 360/67 in double precision.

APPENDIX. THE MODIFIED GIVENS TRANSFORMATION

The Givens transformation which eliminates z_1, if $z_1 \neq 0$, is

$$GW = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{bmatrix} w_1 \cdots w_N \\ z_1 \cdots z_N \end{bmatrix}, \tag{A1}$$

where $c = w_1/r$, $s = z_1/r$, $r = \pm (w_1^2 + z_1^2)^{1/2}$. This requires $\sim 4N$ floating-point multiplications, $2N$ floating-point additions, and one square root. Gentleman [9] has reported on a modification to the Givens transformation which reduces this operation count. Gentleman's idea is presented here in a slightly different form from that found in his paper.

Suppose that W in eq. (A1) is available in factored form:

$$W = D^{1/2}X \equiv \begin{bmatrix} d_1^{1/2} & 0 \\ 0 & d_2^{1/2} \end{bmatrix} \begin{bmatrix} x_1 \cdots x_N \\ y_1 \cdots y_N \end{bmatrix}. \tag{A2}$$

Substituting $D^{1/2}X$ for W and refactoring $GD^{1/2}$ as $\tilde{D}^{1/2}H$ yields

$$GW = GD^{1/2}X = \tilde{D}^{1/2}HX \equiv \begin{bmatrix} \tilde{d}_1^{1/2} & 0 \\ 0 & \tilde{d}_1^{1/2} \end{bmatrix} HX. \tag{A3}$$

The right-hand side of eq. (A3) yields an updated factored form for the matrix product GW. The crucial point is that the matrix H is selected so that two elements are exactly units. This eliminates $2N$ floating-point multiplications when forming the matrix product HX. To preserve numerical stability two cases are considered:

For $|s| < |c|$,

$$\begin{aligned} GD^{1/2} &= \begin{bmatrix} d_1^{1/2}c & d_2^{1/2}s \\ -d_1^{1/2}s & d_2^{1/2}c \end{bmatrix} = \begin{bmatrix} d_1^{1/2}c & 0 \\ 0 & d_2^{1/2}c \end{bmatrix} \begin{bmatrix} 1 & t(d_2/d_1)^{1/2} \\ -t(d_1/d_2)^{1/2} & 1 \end{bmatrix} \\ &= \begin{bmatrix} \tilde{d}_1^{1/2} & 0 \\ 0 & \tilde{d}_2^{1/2} \end{bmatrix} \begin{bmatrix} 1 & d_2y_1/d_1x_1 \\ -y_1/x_1 & 1 \end{bmatrix} \equiv \tilde{D}^{1/2}H, \end{aligned} \tag{A4}$$

where $t = s/c$.

For $|c| \leq |s|$, by similar manipulations,

$$GD^{1/2} = \begin{bmatrix} \tilde{d}_1^{1/2} & 0 \\ 0 & \tilde{d}_2^{1/2} \end{bmatrix} \begin{bmatrix} d_1x_1/d_2y_1 & 1 \\ -1 & x_1/y_1 \end{bmatrix} \equiv \tilde{D}^{1/2}H, \tag{A5}$$

where $\tilde{d}_1^{1/2} = d_2^{1/2}s$ and $\tilde{d}_2^{1/2} = d_1^{1/2}s$. This factorization can be done for any plane rotation matrix.

Only the squares of the scale factors $d_i^{1/2}$ are involved in the nonunit elements of the matrix H defined in eqs. (A4) and (A5), which permits the Givens transformation, eq. (A1), to be computed without square roots. Using the identity $c^2 = (1 + t^2)^{-1}$ and eq. (A4) allows the squares of the scale factors to be updated: $\tilde{d}_i = d_i(1 + t^2)^{-1}$, $i = 1, 2$. Letting $\tau = c/s$ in eq. (A5), we have $\tilde{d}_1 = d_2(1 + \tau^2)^{-1}$ and $\tilde{d}_2 = d_1(1 + \tau^2)^{-1}$.

For $|c| > |s|$ or, equivalently, $|d_1x_1^2| > |d_2y_1^2|$,

$$h_{11} = 1, \qquad h_{12} = d_2y_1/d_1x_1, \tag{A6a}$$

$$h_{21} = -y_1/x_1, \qquad h_{22} = 1, \tag{A6b}$$

$$u = 1 - h_{21}h_{12}, \tag{A6c}$$

$$d_1 := d_1/u, \tag{A6d}$$

$$d_2 := d_2/u, \tag{A6e}$$

$$x_1 := x_1u. \tag{A6f}$$

For $|c| \leq |s|$ or, equivalently, $|d_1x_1^2| \leq |d_2y_1^2|$,

$$h_{11} = d_1x_1/d_2y_1, \qquad h_{12} = 1, \tag{A7a}$$

$$h_{21} = -1, \qquad h_{22} = x_1/y_1, \tag{A7b}$$

$$u = 1 + h_{11}h_{22}, \tag{A7c}$$

$$v = d_1/u, \tag{A7d}$$

$$d_1 := d_2/u \tag{A7e}$$

$$d_2 := v, \tag{A7f}$$

$$x_1 := y_1 u. \tag{A7g}$$

When using the modified Givens transformation in the context of "row accumulation," $d_i > 0$, $i = 1, 2$, the values of u in eqs. (A6) and (A7) will satisfy $1 \leq u \leq 2$. Thus the squares d_i, $i = 1, 2$, decrease by as much as $\frac{1}{2}$ at each updating step. If no rescaling action is taken, these scale factors would ultimately underflow. The details concerning rescaling are implemented in the modified Givens subprograms.

Since only the squares of the weights, i.e. d_1 and d_2, appear in the formulas of eqs. (A6) and (A7), it is possible to use the same formulas to remove a row from a least squares problem simply by setting $d_2 = -1$. Remarks about this row removal method are found in [14, ch. 27].

When the modified Givens transformation is used in the context of the "row removal method" mentioned above, the values of u in eqs. (A6) and (A7) satisfy $0 \leq u \leq 1$. The case $u = 0$ is eliminated by restricting $d_1 \geq 0$. If $d_1 < 0$, we define H as the zero matrix, the updated $d_i = 0$, $i = 1, 2$, and $x_1 = 0$. With this restriction, we have $0 < u \leq 2$ in eqs. (A6) and (A7). Thus the change in the scale factors d_i, $i = 1, 2$, is unbounded at each step. Either underflow or overflow can occur if no rescaling is performed.

The problem is rescaled by the modified Givens subprograms to keep within the conservative limits:

$$\gamma^{-2} \leq |d_i| \leq \gamma^2, \qquad i = 1, 2, \quad \gamma = 4096.$$

Note that when we rescale $d_i := d_i\gamma^2$, we must rescale $h_{ij} := h_{ij}\gamma^{-1}$, $j = 1, 2$, and, when $i = 1$, rescale $x_1 := x_1\gamma^{-1}$.

ACKNOWLEDGMENTS

We are grateful for the contributions that numerous people have made to this project. The Master Test Program was developed by Lawson, with a few modifications by Hanson. The Fortran versions of the BLAS subprograms were written by Lawson, Krogh, Hanson, and J. Dongarra. The assembly coded versions for the Univac 1108 were programmed by Krogh and S. Singletary Gold. The assembly coded versions for the IBM 360/67 were programmed by Hanson and K. Haskell. The assembly coded versions for the CDC 6600 were programmed by Kincaid, J. Sullivan, and E. Williams. Four of these routines were recoded by Hanson and C. Moler. Test runs were made on a variety of machines by P. Fox and E.W. McMahon (Honeywell 6000), P. Knowlton (PDP 10), L. Fosdick (CDC 6600), C. Moler (IBM 360/67), K. Fong (CDC 7600), B. Garbow and J. Dongarra (IBM 370/195), W. Brainerd (Burroughs 6700), and others.

Helpful suggestions, based on previous similar work of their own, were given by P.S. Jensen and C. Bailey. Valuable help was also contributed by J. Wisniewski, W. MacGregor, and G. Terrell.

J. Dongarra supplied versions of several Fortran implementations of the subprograms. The choice of coding technique used by J. Dongarra is based on a

set of tests that was carried out at over 40 different installations with various machines in operation. The choice of coding technique was made on the basis of superior timing performance at the largest number of these sites [8].

REFERENCES

1. ANSI FORTRAN X3.9-1966. American National Standards Institute, New York, 1966.
2. ANSI FORTRAN X3.9-1978. American National Standards Institute, New York, 1978. (Also known as FORTRAN 77.)
3. ANSI Subcommittee X3J3. Clarification of FORTRAN standards—second report. *Comm. ACM 14,* 10 (1971), 628–642.
4. ASA Sectional Committee X3. FORTRAN vs. Basic FORTRAN *Comm. ACM 7,* 10 (Oct. 1964), 591–625.
5. BLUE, J.L. A portable Fortran program to find the Euclidean norm of a vector. *ACM Trans. Math. Software 4,* 1 (March 1978), 15–23.
6. BRENT, R. A Fortran multiple precision arithmetic package. *ACM Trans. Math. Software 4,* 1 (March 1978), 57–70.
7. CODY, W.J. Software for the elementary functions. In *Mathematical Software,* J.R. Rice, Ed., Academic Press, New York, 1971, pp. 171–186.
8. DONGARRA, J.J. Fortran BLAS timing. LINPACK Working Note 3, Argonne Nat. Lab., Argonne, Ill. draft of March 1977.
9. GENTLEMAN, W.M. Least squares computations by Givens transformations without square roots. *J. Inst. Math. Appl. 12* (1973), 329–336.
10. HAMMARLING, S. A note on modifications of the Givens plane rotation. *J. Inst. Math. Appl. 13,* 2 (1974), 215–218.
11. HANSON, R.J., KROGH, F.T., AND LAWSON, C.L. A proposal for standard linear algebra subprograms. TM 33-660, Jet Propulsion Lab., Pasadena, Calif., Nov. 1973.
12. HANSON, R.J., LAWSON, C.L., KINCAID, D.R., AND KROGH, F.T. Basic linear algebra subprograms for FORTRAN usage—an extended report. Sandia Tech. Rep. SAND 77-0898, Sandia Lab., Albuquerque, N. Mex., 1977.
13. LAWSON, C.L. Standardization of FORTRAN callable subprograms for basic linear algebra. Software II, Purdue U., W. Lafayette, Ind., May 1974. (Abstract on p. 261.)
14. LAWSON, C.L., AND HANSON, R.J. *Solving Least Squares Problems.* Prentice-Hall, Englewood, Cliffs, N.J., 1974.
15. STEWART, G.W. The economical storage of plane rotations. *Numer. Math. 25,* 2 (1976), 137–139.

Received July 1977; revised February 1978

ALGORITHM 539
Basic Linear Algebra Subprograms for Fortran Usage [F1]

C. L. LAWSON
Jet Propulsion Laboratory
R. J. HANSON
Sandia Laboratories, Albuquerque
D. R. KINCAID
The University of Texas, Austin
and
F. T. KROGH
Jet Propulsion Laboratory

Key Words and Phrases: linear algebra, utilities
CR Categories: 4.49, 5.14
Language: Fortran, assembly language

DESCRIPTION

This package complements [1], where further details are given.

REFERENCES

1. LAWSON, C.L., HANSON, R.J., KINCAID, D.R., AND KROGH, F.T. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software 5*, 3 (September 1979), 308–323.

ALGORITHM

[Summary information and part of the listing is printed here. The complete listing is available from the ACM Algorithms Distribution Service (see inside back cover for order form), or may be found in "Collected Algorithms from ACM."]

Received 13 July 1977.

The work of the first and fourth authors was supported by the National Aeronautics and Space Administration under Contract NAS 7-100. The work of the second author was supported by the U.S. Energy Research and Development Administration (ERDA) under Contract AT (29-1)-789 and (at Washington State University) by the Office of Naval Research under Contract NR 044-457.
Authors' addresses: C.L. Lawson, Jet Propulsion Laboratory, M/S 125-128, 4800 Oak Grove Drive, Pasadena, CA 91103; R.J. Hanson, Numerical Mathematics, Div. 5122, Sandia Laboratories, Albuquerque, NM 87115; D.R. Kincaid, Center for Numerical Analysis, The University of Texas at Austin, Austin, TX 78712; F.T. Krogh, Jet Propulsion Laboratory, M/S 125-128, 4800 Oak Grove Drive, Pasadena, CA 91103.

Contents: This package consists of four files. The first file contains Fortran versions for the BLAS (38 subprograms), programs for testing the BLAS (13 modules), and 18 subprograms from Brent's multiple precision package that are used in the implementation of the extended precision inner products. The remaining 3 files contain Fortran callable assembly language versions for 3 different machines: IBM 360/370 series, CDC 6000 series, and Univac 1100 series.

```
      REAL FUNCTION SDOT(N,SX,INCX,SY,INCY)                                  1Ø
C                                                                            2Ø
C     RETURNS THE DOT PRODUCT OF SINGLE PRECISION SX AND SY.                 3Ø
C     SDOT = SUM FOR I = Ø TO N-1 OF  SX(LX+I*INCX) * SY(LY+I*INCY),         4Ø
C     WHERE LX = 1 IF INCX .GE. Ø, ELSE LX = (-INCX)*N, AND LY IS            5Ø
C     DEFINED IN A SIMILAR WAY USING INCY.                                   6Ø
C                                                                            7Ø
      REAL SX(1),SY(1)                                                       8Ø
      SDOT = Ø.ØEØ                                                           9Ø
      IF(N.LE.Ø)RETURN                                                      1ØØ
      IF(INCX.EQ.INCY) IF(INCX-1)5,2Ø,6Ø                                    11Ø
    5 CONTINUE                                                              12Ø
C                                                                           13Ø
C        CODE FOR UNEQUAL INCREMENTS OR NONPOSITIVE INCREMENTS.             14Ø
C                                                                           15Ø
      IX = 1                                                                16Ø
      IY = 1                                                                17Ø
      IF(INCX.LT.Ø)IX = (-N+1)*INCX + 1                                     18Ø
      IF(INCY.LT.Ø)IY = (-N+1)*INCY + 1                                     19Ø
      DO 1Ø I = 1,N                                                         2ØØ
        SDOT = SDOT + SX(IX)*SY(IY)                                         21Ø
        IX = IX + INCX                                                      22Ø
        IY = IY + INCY                                                      23Ø
   1Ø CONTINUE                                                              24Ø
      RETURN                                                                25Ø
C                                                                           26Ø
C        CODE FOR BOTH INCREMENTS EQUAL TO 1                                27Ø
C                                                                           28Ø
C                                                                           29Ø
C        CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 5.       3ØØ
C                                                                           31Ø
   2Ø M = MOD(N,5)                                                          32Ø
      IF( M .EQ. Ø ) GO TO 4Ø                                               33Ø
      DO 3Ø I = 1,M                                                         34Ø
        SDOT = SDOT + SX(I)*SY(I)                                           35Ø
   3Ø CONTINUE                                                              36Ø
      IF( N .LT. 5 ) RETURN                                                 37Ø
   4Ø MP1 = M + 1                                                           38Ø
      DO 5Ø I = MP1,N,5                                                     39Ø
        SDOT = SDOT + SX(I)*SY(I) + SX(I + 1)*SY(I + 1) +                   4ØØ
     $   SX(I + 2)*SY(I + 2) + SX(I + 3)*SY(I + 3) + SX(I + 4)*SY(I + 4)    41Ø
   5Ø CONTINUE                                                              42Ø
      RETURN                                                                43Ø
C                                                                           44Ø
C        CODE FOR POSITIVE EQUAL INCREMENTS .NE.1.                          45Ø
C                                                                           46Ø
   6Ø CONTINUE                                                              47Ø
      NS=N*INCX                                                             48Ø
      DO 7Ø I=1,NS,INCX                                                     49Ø
        SDOT = SDOT + SX(I)*SY(I)                                           5ØØ
   7Ø   CONTINUE                                                            51Ø
      RETURN                                                                52Ø
      END                                                                   53Ø
```