.

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

An Extended Set of Fortran Basic Linear Algebra Subprograms

Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson

Mathematics and Computer Science Division

Technical Memorandum No. 41 (Revision 3)

November, 1986

.

ABSTRACT

This paper describes an extension to the set of Basic Linear Algebra Subprograms. The extensions are targeted at matrix-vector operations which should provide for efficient and portable implementations of algorithms for high-performance computers.

An Extended Set of Fortran
Basic Linear Algebra Subprograms

Jack J. Dongarra †

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439

Jeremy Du Croz

Numerical Algorithms Group Ltd.
NAG Central Office, Mayfield House
256 Banbury Road, Oxford OX2 7DE

Sven Hammarling

Numerical Algorithms Group Ltd.
NAG Central Office, Mayfield House
256 Banbury Road, Oxford OX2 7DE

Richard J. Hanson ‡

Applied Mathematics Division 2646
Sandia National Laboratory
Albuquerque, New Mexico 87185

Abstract — This paper describes an extension to the set of Basic Linear Algebra Subprograms. The extensions are targeted at matrix-vector operations which should provide for efficient and portable implementations of algorithms for high-performance computers.

1. Introduction

In 1973 Hanson, Krogh, and Lawson [9] described the advantages of adopting a set of basic routines for problems in linear algebra. The original basic linear algebra subprograms, now commonly referred to as the BLAS and fully described in Lawson, Hanson, Kincaid, and Krogh [10,11], have been very successful and have been used in a wide range of software including LINPACK [4] and many of the algorithms published by the ACM Transactions on Mathematical Software. In particular they are an aid to clarity, portability, modularity and maintenance of software and they have become a *de facto* standard for the elementary vector operations. An excellent discussion of the *raison d' être* of the BLAS is given in Dodson and Lewis [1].

---

† Work supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.

‡ Work supported by the U.S. Department of Energy.

Typeset on April 18, 1988.

Special versions of the BLAS, in some cases machine code versions, have been implemented on a number of computers, thus improving the efficiency of the BLAS. However, with some of the modern machine architectures, the use of the BLAS is not the best way to improve the efficiency of higher level codes. On vector machines, for example, one needs to optimize at least at the level of matrix-vector operations in order to approach the potential efficiency of the machine (see [2,3]); the use of the BLAS inhibits this optimization because they hide the matrix-vector nature of the operations from the compiler.

We believe that the time is right to specify an additional set of BLAS designed for matrix-vector operations. It has been our experience that a small set of matrix-vector operations occur frequently in the implementation of many of the most common algorithms in linear algebra. We define here the basic operations for that set, together with the naming conventions and the calling sequences. Routines at this level should provide a reasonable compromise between the sometimes conflicting aims of efficiency and modularity and it is our hope that efficient implementations will become available on a wide range of computer architectures.

During the Gatlinburg meeting of June, 1984 (Waterloo, Ontario) discussions among the participants encouraged us to prepare a proposed set of Level 2 BLAS. At about the same time IFIP Working Group 2.5 started a project on the same subject at their annual meeting in Pasadena, CA.

An initial proposal was drafted and presented at the Parvec IV workshop held at Purdue University, Oct. 29-30, 1984. A series of meetings was planned so that the project would reflect the best thinking of the mathematical software community. Four meetings soliciting input were held. These occurred at SIAM conferences: The Spring Meeting of the Society, (Seattle, WA, July 16-20, 1984); The Conference on Applied Linear Algebra (Raleigh, NC, April 29-May 2, 1985); The Fall Meeting of the Society, (Tempe, AZ, October 28-30, 1985); The Conference on Parallel Processing for Scientific Computing (Norfolk, VA, November 18-21, 1985). In addition, we had discussions with the ACM SIGNUM Board at the board meeting in Seattle, WA in July 1984.

Earlier, a modified proposal was printed in the SIGNUM Newletter, [6]. In that document we invited readers to send us their views and suggestions for changes to the design of the extended BLAS. Thus we have appealed to a wide audience within the mathematical software community. Our hope is that the proposed set of routines that constitute the extended BLAS will find wide application in future software for numerical linear algebra and provide a useful tool for implementors and users.

In this paper we shall refer to the existing BLAS of Lawson *et al.* as ‘‘Level 1 BLAS’’, and the new extended set as ‘‘Level 2 BLAS’’. The Level 2 BLAS involve $O(mn)$ scalar operations where m and n are the dimensions of the matrix involved. These could be programmed by a series of calls to the Level 1 BLAS, though we do not recommend that they be implemented in that way. Hence, in a natural sense, the Level 2 BLAS are performing basic operations at one level higher than the Level 1 BLAS.

In [7] we present a model implementation of the Level 2 BLAS in Fortran 77 (extended to include a COMPLEX*16 data type), and also a set of rigorous test programs.

2. Scope of the Level 2 BLAS

The following three types of basic operation are performed by the Level 2 BLAS:

a) Matrix-vector products of the form

$$y \leftarrow \alpha Ax + \beta y, \quad y \leftarrow \alpha A^T x + \beta y, \quad \text{and} \quad y \leftarrow \alpha \bar{A}^T x + \beta y$$

where α and β are scalars, x and y are vectors and A is a matrix, and

$$x \leftarrow Tx, \quad x \leftarrow T^T x, \quad \text{and} \quad x \leftarrow \bar{T}^T x,$$

where x is a vector and T is an upper or lower triangular matrix.

b) Rank-one and rank-two updates of the form

$$A \leftarrow \alpha x y^T + A, \quad A \leftarrow \alpha x \bar{y}^T + A, \quad H \leftarrow \alpha x \bar{x}^T + H, \quad \text{and } H \leftarrow \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T + H,$$

where H is a Hermitian matrix.

c) Solution of triangular equations of the form

$$x \leftarrow T^{-1} x, \; x \leftarrow T^{-T} x, \quad \text{and} \quad x \leftarrow \bar{T}^{-T} x,$$

where T is a non-singular upper or lower triangular matrix.

Where appropriate, the operations are applied to general, general band, Hermitian, Hermitian band, triangular, and triangular band matrices in both real and complex arithmetic, and in single and double precision.

In Appendix B we propose corresponding sets of routines in which the internal computation is performed in extended precision, and the vectors x and/or y are stored in extended precision, so that the extra internal precision is not all discarded on return from the routine. This proposal is aimed at environments that support extended precision arithmetic, for example machines performing IEEE arithmetic [12]. We propose these routines as an optional extension to the Level 2 BLAS because it is not possible to specify a complete set within the confines of ANSI Fortran 77. The only case that can be realized within the standard is where the matrix is real single precision and the extended precision vectors are real double precision. Code for these routines is not included in [7].

3. Naming Conventions

The name of a Level 2 BLAS is in the LINPACK style and consists of five characters, the last of which may be blank. The first character in the name denotes the Fortran data type of the matrix, as follows:

S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	COMPLEX*16 or DOUBLE COMPLEX (if available)

Characters two and three in the name denote the kind of matrix involved, as follows:

GE	General matrix
GB	General band matrix
HE	Hermitian matrix
SY	Symmetric matrix
HP	Hermitian matrix stored in packed form
SP	Symmetric matrix stored in packed form
HB	Hermitian band matrix
SB	Symmetric band matrix
TR	Triangular matrix
TP	Triangular matrix stored in packed form
TB	Triangular band matrix

The fourth and fifth characters in the name denote the type of operation, as follows:

MV	- Matrix-vector product
R	- Rank-one update
R2	- Rank-two update
SV	- Solve a system of linear equations

The available combinations are indicated in Table 3.1 below. In the first column, under *complex*, the initial C may be replaced by Z. In the second column, under *real*, the initial S may be replaced by D. See Appendix C for the full subroutine calling sequences.

The collection of routines can be thought of as being divided into four separate parts, *real*, *double precision*, *complex*, and *complex*16*. The routines can be written in ANSI standard Fortran 77, with the exception of the routines that use COMPLEX*16 variables. These routines are included for completeness and for their usefulness on those systems which support this data type, but because they do not conform to the Fortran standard, they may not be available on all machines.

Table 3.1

complex	*real*	MV	R	R2	SV
CGE	SGE	*	*		
CGB	SGB	*			
CHE	SSY	*	*	*	
CHP	SSP	*	*	*	
CHB	SSB	*			
CTR	STR	*			*
CTP	STP	*			*
CTB	STB	*			*

For the general rank-1 update (GER) we specify two complex routines: CGERC for $A \leftarrow \alpha x \bar{y}^T + A$ and CGERU for $A \leftarrow \alpha x y^T + A$. This is the only exception to the one to one correspondence between real and complex routines. See section 7 for further discussion.

We do not specify routines for rank-one and rank-two updates applied to band matrices because these can be obtained by calls to the rank-one and rank-two full matrix routines. This is illustrated in Appendix A.

4. Argument Conventions

We follow a convention for the argument lists similar to that for the Level 1 BLAS, but with extensions where comparable arguments are not present in the Level 1 BLAS. The order of arguments is as follows:

a) Arguments specifying options.
b) Arguments defining the size of the matrix.
c) Input scalar.
d) Description of the input matrix.
e) Description of input vector(s).
f) Input scalar (associated with input-output vector).
g) Description of the input-output vector.
h) Description of the input-output matrix.

Note that not each category is present in each of the routines.

The arguments that specify options are character arguments with the names TRANS, UPLO, and DIAG. TRANS is used by the matrix-vector product routines as follows:

Value	Meaning
‘N’	Operate with the matrix.
‘T’	Operate with the transpose of the matrix.
‘C’	Operate with the conjugate transpose of the matrix.

In the real case the values ‘T’ and ‘C’ have the same meaning.

UPLO is used by the Hermitian, symmetric, and triangular matrix routines to specify whether the upper or lower triangle is being referenced as follows:

Value	Meaning
‘U’	Upper triangle
‘L’	Lower triangle

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

Value	Meaning
‘U’	Unit triangular
‘N’	Non-unit triangular

When DIAG is supplied as ‘U’ the diagonal elements are not referenced.

We recommend that the corresponding lower case characters be accepted with the same meaning, although, because they are not included in the standard Fortran character set, their use may not be supported on all systems. See Section 7 for further discussion.

It is worth noting that actual character arguments in Fortran may be longer than the corresponding dummy arguments. Therefore, for example, the value ‘T’ for TRANS may be passed as ‘TRANSPOSE’.

The size of the matrix is determined by the arguments M and N for an m by n rectangular matrix, and by the argument N for an n by n symmetric, Hermitian, or triangular matrix. Note that it is permissible to call the routines with M or N = 0, in which case the routines exit immediately without referencing their vector or matrix arguments. The bandwidth is determined by the arguments KL and KU for a rectangular matrix with kl sub-diagonals and ku super-diagonals; and by the argument K for a symmetric, Hermitian, or triangular matrix with k sub-diagonals and/or super-diagonals.

The description of the matrix consists of either the array name (A) followed by the leading dimension of the array (LDA) as declared in the calling (sub) program, when the matrix is being stored in a two-dimensional array; or the array name (AP) alone when the matrix is being stored in packed form in a one-dimensional array. In the former case the actual array must contain at least $((n-1)d + l)$ elements, where d is the leading dimension of the array, $d \ge l$, and $l = m$ for the GE routines, $l = n$ for the SY, HE and TR routines, $l = kl + ku + 1$ for the GB routines, and $l = k+1$ for the SB, HB or TB routines. For packed storage the actual array must contain at least $n(n+1)/2$ elements.

The scalars always have the dummy argument names ALPHA and BETA.

As with the existing BLAS the description of a vector consists of the name of the array (X or Y) followed by the storage spacing (increment) in the array of the vector elements (INCX or INCY). The increment is allowed to be positive or negative, but not zero (see section 7). When the vector x consists of k elements, then the corresponding actual array argument X must be of length at least $(1 + (k-1) \mid INCX \mid)$. For those routines that include the argument BETA, when BETA is supplied as zero then the array Y need not be set on input, so that operations such as $y \leftarrow \alpha Ax$ may be performed without initially setting y to zero.

Note that the MV routines must not be called with the dummy arguments X and Y associated with the same actual argument. (Such association is not supported by the model implementation, and would also contravene the Fortran standard.)

The following values of arguments are invalid:

Any value of the character arguments DIAG, TRANS, or UPLO
whose meaning is not specified.
M < 0
N < 0
KL < 0
KU < 0
K < 0
LDA < max(1, M) for the GE routines.
LDA < KL + KU + 1 for the GB routines.
LDA < N for the HE, SY, and TR routines.
LDA < K + 1 for the HB, SB, and TB routines.
INCX = 0
INCY = 0

If a routine is called with an invalid value for any of its arguments, then it must report the fact and terminate execution of the program. In the model implementation (see[7]), each routine, on detecting an error, calls a common error handling routine XERBLA, passing to it the name of the routine and the number of the first argument which is in error. Specialized implementations may call system-specific exception-handling and diagnostic facilities, either via an auxiliary routine XERBLA or directly from the routines. One advantage of using XERBLA is that the test program can then test that all errors are detected (see [7]).

5. Storage Conventions

Unless otherwise stated it is assumed that matrices are stored conventionally in a 2-dimensional array with matrix-element a_{ij} stored in array-element A(I,J).

The routines for real symmetric and complex Hermitian matrices allow for the matrix either to be stored in the upper (UPLO = ‘U’) or lower triangle (UPLO = ‘L’) of a two dimensional array, or to be packed in a one dimensional array. In the latter case the upper triangle may be packed sequentially column by column (UPLO = ‘U’), or the lower triangle may be packed sequentially column by column (UPLO = ‘L’). Note that for real symmetric matrices packing the upper triangle by columns is equivalent to packing the lower triangle by rows, and packing the lower triangle by columns is equivalent to packing the upper triangle by rows. (For complex Hermitian matrices the only difference is that the off-diagonal elements are conjugated.)

For triangular matrices the argument UPLO serves to define whether the matrix is upper (UPLO = ‘U’) or lower (UPLO = ‘L’) triangular. In packed storage the triangle has to be packed by column.

The band matrix routines allow storage in the same style as with LINPACK, so that the j^{th} column of the matrix is stored in the j^{th} column of the Fortran array. For a general band matrix the diagonal of the matrix is stored in row $ku+1$ of the array. For a Hermitian or symmetric matrix either the upper triangle (UPLO = ‘U’) may be stored in which case the leading diagonal is in row $k+1$ of the array, or the lower triangle (UPLO = ‘L’) may be stored in which case the leading diagonal is in the first row of the array. For an upper triangular band matrix (UPLO = ‘U’) the leading diagonal is in row $k+1$ of the array and for a lower triangular band matrix (UPLO = ‘L’) the leading diagonal is in the first row.

For a Hermitian matrix the imaginary parts of the diagonal elements are of course zero and thus the imaginary parts of the corresponding Fortran array elements need not be set, but are assumed to be zero. In the R and R2 routines these imaginary parts will be set to zero on return, except when $\alpha = 0$, in which case the routines exit immediately.

For packed triangular matrices the same storage layout is used whether or not DIAG = ‘U’, i.e. space is left for the diagonal elements even if those array elements are not referenced.

6. Specification of the Level 2 BLAS

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

```
INTEGER      INCX, INCY, K, KL, KU, LDA, M, N
CHARACTER*1 DIAG, TRANS, UPLO
```

For routines whose first letter is an S:

```
REAL   ALPHA, BETA
```

```
REAL  X(*), Y(*)
REAL  A(LDA,*)
REAL  AP(*)
```

For routines whose first letter is a D

```
DOUBLE PRECISION  ALPHA, BETA
DOUBLE PRECISION  X(*), Y(*)
DOUBLE PRECISION  A(LDA,*)
DOUBLE PRECISION  AP(*)
```

For routines whose first letter is a C:

```
COMPLEX  ALPHA
COMPLEX  BETA
COMPLEX  X(*), Y(*)
COMPLEX  A(LDA,*)
COMPLEX  AP(*)
```

except that for CHER and CHPR the scalar α is real so that the first declaration above is replaced by:

```
REAL  ALPHA
```

For routines whose first letter is Z one of the alternative (non-standard) specifications:

```
COMPLEX*16  ALPHA              DOUBLE COMPLEX  ALPHA
COMPLEX*16  BETA               DOUBLE COMPLEX  BETA
COMPLEX*16  X(*), Y(*)         DOUBLE COMPLEX  X(*), Y(*)
COMPLEX*16  A(LDA,*)           DOUBLE COMPLEX  A(LDA,*)
COMPLEX*16  AP(*)              DOUBLE COMPLEX  AP(*)
```

except that for ZHER and ZHPR the first declaration above is replaced by:

```
DOUBLE PRECISION  ALPHA
```

The generic names, argument lists and specifications for the extended BLAS now follow. Refer to table 3.1 for the specific subroutine names.

In the following argument lists the array name AP is used for the cases where the matrix A is supplied in packed form; in all other cases the array name A is used.

a) General Matrix Vector Products

for a general matrix:
_GEMV (TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

for a general band matrix:
_GBMV (TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

Operation:

if TRANS = ‘N’, $y \leftarrow \alpha A x + \beta y$

if TRANS = ‘T’, $y \leftarrow \alpha A^T x + \beta y$

if TRANS = ‘C’, $y \leftarrow \alpha \bar{A}^T x + \beta y$.

b) Symmetric or Hermitian Matrix Vector Products

for a symmetric or Hermitian matrix:
_SYMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
_HEMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

for a symmetric or Hermitian matrix in packed storage:
_SPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
_HPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)

for a symmetric or Hermitian band matrix:
_SBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
_HBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

Operation:

$$y \leftarrow \alpha A x + \beta y \ .$$

c) Triangular Matrix Vector Products

for a triangular matrix:
_TRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

for a triangular matrix in packed storage:
_TPMV (UPLO, TRANS, DIAG, N, AP, X, INCX)

for a triangular band matrix:
_TBMV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)

Operation:

if TRANS = 'N', $x \leftarrow Ax$

if TRANS = 'T', $x \leftarrow A^T x$

if TRANS = 'C', $x \leftarrow \bar{A}^T x$

d) Triangular equation solvers

for a triangular matrix:
_TRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

for a triangular matrix in packed storage:
_TPSV (UPLO, TRANS, DIAG, N, AP, X, INCX)

for a triangular band matrix:
_TBSV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)

Operation:

if TRANS = 'N', $x \leftarrow A^{-1} x$

if TRANS = 'T', $x \leftarrow A^{-T} x$

if TRANS = 'C', $x \leftarrow \bar{A}^{-T} x$.

e) General Rank-1 Updates:

for a general matrix:
GER (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)

for real matrices:
SGER or DGER performs the operation $A \leftarrow \alpha x y^T + A$.

for complex matrices:

CGERC or ZGERC performs the operation $A \leftarrow \alpha x \bar{y}^T + A$
and
CGERU or ZGERU performs the operation $A \leftarrow \alpha x y^T + A$.

f) Symmetric or Hermitian Rank-1 Updates:

for a symmetric or Hermitian matrix:
_SYR (UPLO, N, ALPHA, X, INCX, A, LDA)
_HER (UPLO, N, ALPHA, X, INCX, A, LDA)

for symmetric or Hermitian matrix in packed storage:
_SPR (UPLO, N, ALPHA, X, INCX, AP)
_HPR (UPLO, N, ALPHA, X, INCX, AP)

Operation:

$$A \leftarrow \alpha x \bar{x}^T + A$$

for real symmetric matrices this is simply

$$A \leftarrow \alpha x x^T + A \ .$$

g) Symmetric or Hermitian Rank-2 Updates:

for a symmetric or Hermitian matrix:
_SYR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
_HER2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)

for symmetric or Hermitian matrix in packed storage:
_SPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
_HPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)

Operation:

$$A \leftarrow \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T + A$$

for real symmetric matrices this is simply

$$A \leftarrow \alpha x y^T + \alpha y x^T + A \ .$$

7. Rationale

The three basic matrix-vector operations chosen (Section 2) were obvious candidates because they occur in a wide range of linear algebra applications, and they occur at the innermost level of many algorithms. The hard decision was to restrict the scope only to these operations, since there are many other potential candidates, such as matrix scaling and sequences of plane rotations. Similarly, we could have extended the scope by applying the operations to other types of matrices such as complex symmetric or augmented band matrices. We have aimed at a reasonable compromise between a much larger number of routines each performing one type of operation (e.g. $x \leftarrow L^{-T}x$), and a smaller number of routines with a more complicated set of options. There are in fact, in each precision, 16 real routines performing altogether 43 different operations, and 17 complex routines performing 58 different operations.

We feel that to extend the scope further would significantly reduce the chances of having the routines implemented efficiently over a wide range of machines, because it would place too heavy a burden on implementors. On the other hand, to restrict the scope further would place too narrow a limit on the potential applications of the Level 2 BLAS.

We have adhered to the conventions of the Level 1 BLAS in allowing an increment argument to be associated with each vector, so that a vector could, for example, be a row of a matrix. This increment may be negative, in which case the elements of the vectors are taken in reverse order. This affects the definition of the operation. For example, if $m = n = 3$ and INCX and INCY are both negative, the _GEMV routines with TRANS='N' perform the operation:

$$\begin{pmatrix} y_3 \\ y_2 \\ y_1 \end{pmatrix} \leftarrow \alpha \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} + \beta \begin{pmatrix} y_3 \\ y_2 \\ y_1 \end{pmatrix}.$$

However, in contrast to the Level 1 BLAS we do not allow INCX or INCY to be zero. This feature would have little usefulness, would complicate implementation of the routines on many vector machines, and, when the associated vector is an output vector, its meaning is ambiguous.

As noted earlier, corresponding to the real routine SGER we specify two complex routines CGERC (for $A \leftarrow \alpha x \bar{y}^T + A$) and CGERU (for $A \leftarrow \alpha x y^T + A$). Both are frequently required. An alternative would be to provide a single complex routine CGER with an option argument; however this argument would have become redundant in the real routine SGER. Rather than have redundant arguments, or different argument lists for the real and complex routines, we have chosen two distinct complex routines; they are analogous to the Level 1 BLAS CDOTC ($c \leftarrow \bar{x}^T y$) and CDOTU ($c \leftarrow x^T y$).

Note that no check has been included for singularity, or near singularity, in the triangular equation solving routines. The requirements for such a test depend upon the application and so we felt that this should not be included, but should instead be performed before calling the triangular solver.

On certain machines, which do not use the ASCII sequence on all of their Fortran systems, lower case characters may not exist, so that the innocent looking argument '*t*', passed through the argument TRANS for designating a transposed matrix, is not in the Fortran character set. Some UNIVAC systems do not have a lower case representation using the 'field data' character set. On the CDC NOS-2 system, a mechanism is provided for a full 128 ASCII character set by using pairs of 6-bit host characters for certain 7-bit ASCII characters. This means that there is a '2 for 1' physical extension of the logical records that contain lower case letters. This fact can hamper portability of codes written on ASCII machines that are later moved to CDC systems. The only safe way to proceed is to convert the transported text entirely into the Fortran character set. On the other hand we believe that users on ASCII character set systems may wish to treat upper and lower case letters as equivalent in meaning. If this is done, it means that text that will be transported to machines of unknown types must have the ASCII set mapped into the Fortran character set before the text is moved.

The band storage scheme used by the GB, HB, SB, and TB routines has columns of the matrix stored in columns of the array, and diagonals of the matrix stored in rows of the array. This is the storage scheme used by LINPACK. An alternative scheme (used in some EISPACK [8,11] routines) has rows of the matrix stored in rows of the array, and diagonals of the matrix stored in columns of the array. The latter scheme has the advantage that a band matrix-vector product of the form $y \leftarrow \alpha Ax + \beta y$ can be computed using long vectors (the diagonals of the matrix) stored in contiguous elements, and hence is much more efficient on some machines (e.g. CDC Cyber 205) than the first scheme. However other computations involving band matrices, such as $x \leftarrow Tx$, $x \leftarrow T^{-1}x$ and LU and U^TU factorization, cannot be organized 'by diagonals'; instead the computation sweeps along the band, and the LINPACK storage scheme has the advantage of reducing the number of page swaps and allowing contiguous vectors (the columns of the matrix) to be used.

We considered the possibility of generalizing the rank-1 and rank-2 updates to rank-k updates. Rank-k updates with $k > 1$ (but $k \ll n$) can achieve significantly better performance on some machines than rank-1 [5]. But to take advantage of this usually requires complicating the calling algorithm; and moreover rank-k updates with $k \approx n$ would allow an even higher level operation such as matrix multiplication 'in by the back door'. We prefer to keep to a clean concept of genuine matrix-vector operations.

8. Acknowledgements

A draft proposal that led to this specification was discussed at the Parvec IV Workshop organized by John Rice at Purdue University on October 29-30, 1984 and at various SIAM conferences. We wish to thank all the participants at the workshop and meetings for their comments, discussions, and encouragement, as well as the many people who have sent us comments separately. We particulary thank Ingrid Bucher and Tom Jordan who proposed the use of the β parameter.

13. References

[1] D.S. Dodson and J.G. Lewis, "Issues relating to extension of the Basic Linear Algebra Subprograms", ACM SIGNUM Newsletter, vol 20, no 1, (1985), 19-22.

[2] J.J. Dongarra and S.C. Eisenstat, ‘‘Squeezing the Most out of an Algorithm in CRAY Fortran,’’ *ACM Transactions on Mathematical Software,* Vol. 10, No. 3, (1984), 221-230.

[3] J.J. Dongarra, *Increasing the Performance of Mathematical Software through High-Level Modularity.* Proceedings of the Sixth International Symposium on Computing Methods in Engineering and Applied Sciences. (Versailles, France). North-Holland (1984), pp 239-248.

[4] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart, *LINPACK Users' Guide,* SIAM Publications, Philadelphia, 1979.

[5] J.J. Dongarra, L. Kaufman, and S. Hammarling, *Squeezing the Most out of Eigenvalue Solvers on High-Performance Computers,* Linear Algebra and Its Applications, 77, pp 113-136, (1986).

[6] J.J. Dongarra, J.J. Du Croz, S. Hammarling, R.J. Hanson, *A Proposal for an Extended Set of Fortran Basic Linear Algebra Subprograms,* ACM SIGNUM Newsletter, vol. 20, no. 1, 1985.

[7] J.J. Dongarra, J.J. Du Croz, S. Hammarling, R.J. Hanson, *An Extended Set of Basic Linear Algebra Subprograms: Model Implementation and Test Programs,* Argonne National Laboratory Report, ANL MCS-TM 81, January 1987.

[8] B.S. Garbow, J.M. Boyle, J.J. Dongarra, C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide Extension*, Lecture Notes in Computer Science, Vol. 51, Springer-Verlag, Berlin, 1977.

[9] R. Hanson, F. Krogh, and C. Lawson, *A Proposal for Standard Linear Algebra Subprograms,* SIGNUM Newsletter, 8, (1973), 16.

[10] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ‘‘Basic Linear Algebra Subprograms for Fortran Usage,’’ *ACM Transactions on Mathematical Software* **5** (1979), 308-323.

[11] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ‘‘Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage,’’ *ACM Transactions on Mathematical Software* **5** (1979), 324-325.

[12] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985, The IEEE Inc., New York, (1985).

Appendix A

In this appendix we illustrate how to use the full matrix update routines to obtain rank-one and rank-two updates to band matrices. We assume that the vectors x and y are such that no fill-in occurs outside the band, in which case the update affects only a full rectangle within the band matrix A. This is illustrated in Fig. A.1 for the case where $m=n=9$, $kl=2$, $ku=3$ and the update commences in row (and column) $l=3$.

```
(  *   *   *   *                       )   ( 0 ) ( 0   0   *   *   *   *   0   0   0 )
(  *   *   *__*__*__                   )   ( 0 )
(  *   *  |*   *   *   *|              )   ( * )
(      *  |*   *   *   *|  *           )   ( * )
(         |*__*__*__*|  *   *          ) + ( * )
(              *   *   *   *   *   *   )   ( 0 )
(                  *   *   *   *   *   )   ( 0 )
(                      *   *   *   *   )   ( 0 )
(                          *   *   *   )   ( 0 )
```

$$A + xy^T$$

Figure A.1

We see that the update affects only that part of A indicated by the dotted lines, that is, the $(kl+1)$ by $(ku+1)$ part of A starting at a_{ll}.

The routines that we could have included are _GBR, _SBR, and _SBR2 (in the complex case _HBR and _HBR2). Their argument lists could have been

```
_GBR ( M, N, KL, KU, L, ALPHA, X, INCX, Y, INCY, A, LDA )
_SBR ( UPLO, N, K, L, ALPHA, X, INCX, A, LDA )
_SBR2 ( UPLO, N, K, L, ALPHA, X, INCX, Y, INCY, A, LDA )
```

where the argument L denotes the starting row and column for the update and the elements x_l and y_l, of the vectors x and y, are in elements X(1) and Y(1) of the arrays X and Y.

Calls to SGBR can be achieved by

```
KM = MIN (KL+1, M-L+1)
KN = MIN (KU+1, N-L+1)
CALL SGER (KM, KN, ALPHA, X, INCX, Y, INCY, A(KU+1, L), MAX(KM, LDA-1))
```

Calls to SSBR can be achieved by

```
KN = MIN(K+1, N-L+1)
IF (UPLO .EQ. 'U') THEN
   CALL SSYR('U', KN, ALPHA, X, INCX, A(K+1, L), MAX(1, LDA-1))
ELSE
   CALL SSYR('L', KN, ALPHA, X, INCX, A(1, L), MAX(1, LDA-1))
ENDIF
```

and similarly for calls to SSBR2.

Appendix B

In this appendix we propose an additional set of real and complex Level 2 routines which allow extended precision matrix-vector operations to be performed. The names of these routines are obtained by preceding the character representing the Fortran data type (S or C), by the character E. The matrix is always stored in working precision (which is single precision for the ES- and EC- set of routines, and double precision for the ED- and EZ- set). The computation must be performed in extended precision (which is more accurate than the working precision).

Such routines are useful, for example, in the accurate computation of residuals in iterative refinement. Many machines have extended precision registers in which extended precision computation is performed at little or no extra cost. However, in order to allow the additional precision to be carried through a series of calls to these routines, at least one , in some cases both, of the vectors x and y must be stored in extended precision.

These routines are to perform the operations described in section 2 as follows.

For the matrix-vector operations

$$y \leftarrow \alpha Ax + \beta y, \;\; y \leftarrow \alpha A^T x + \beta y, \;\; y \leftarrow \alpha \bar{A}^T x + \beta y$$

α, β, A, and x are working precision, y is extended precision and the computation of y is to be performed in extended precision.

For the triangular operations

$$x \leftarrow Tx, \;\; x \leftarrow T^T x, \;\; x \leftarrow \bar{T}^T x,$$
$$x \leftarrow T^{-1} x, \;\; x \leftarrow T^{-T} x, \;\; x \leftarrow \bar{T}^{-T} x$$

T is working precision, x is extended precision and the computation of x is to be performed in extended precision.

For the rank-one and rank-two updates

$$A \leftarrow \alpha x y^T + A, \;\; A \leftarrow \alpha x \bar{y}^T + A,$$
$$H \leftarrow \alpha x \bar{x}^T + H, \;\; H \leftarrow \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T + H,$$

α, A, and H are working precision, x and y are extended precision and the computation is to be performed in extended precision.

The precise nature of the extended precision data type cannot be specifed here. The amount of extra precision available will depend on the architecture of the machine and the compiler; whether this is sufficient, will depend upon the application. For the rest of this appendix we assume that double precision is used to implement extended precision in the ES and EC routines, and quadruple precision in the ED and EZ routines (note however that quadruple precision is not specified in IEEE arithmetic). Then the ES- set of routines can be called and implemented in standard Fortran 77. The EC- set require the addition of a COMPLEX*16 data type, as does the basic Z- set, but can be used across a wide range of machines. The ED- set require the addition of a REAL*16 (quadruple precision real) data type, while the EC- set require a COMPLEX*32 (quadruple precision complex) data type; these data types are provided on some systems. We strongly recommend that if implementors provide extended precision routines using these data types, they adhere to the specifications described here, so that at least a limited degree of portability may be achieved.

To test thoroughly that extended precision is used as specified in the internal computations requires an extra degree of sophistication from the test program. For all these reasons, neither a model implementation of the extended precision routines, nor a test program for them, have been included in [7]; code for the ES- and EC- sets of routines may be obtained from the authors.

The specifications of the arguments remain exactly as in Section 6 except the following:

for ESGEMV, ESGBMV, ESSYMV, ESSBMV, ESSPMV, ESGER, ESSYR2 and ESSPR2.

```
DOUBLE PRECISION    Y(*)
```

for the corresponding EC-routines:

```
COMPLEX*16          Y(*)      (or equivalent)
```

for the corresponding ED- routines:

```
REAL*16             Y(*)      (or equivalent)
```

for the corresponding EZ- routines:

```
COMPLEX*32          Y(*)      (or equivalent)
```

for ESTRMV, ESTBMV, ESTPMV, ESTRSV, ESTBSV, ESTPSV, ESGER, ESSYR, ESSPR, ESSYR2 and ESSPR2:

```
DOUBLE PRECISION    X(*)
```

for the corresponding EC- routines:

```
COMPLEX*16      X(*)    (or equivalent)
```

for the corresponding ED- routines:

```
REAL*16         X(*)    (or equivalent)
```

for the corresponding EZ- routines:

```
COMPLEX*32      X(*)    (or equivalent)
```

We thank Velvel Kahan for insisting that we think about the extended precision issue.

Appendix C

This appendix contains the calling sequences for all the Level 2 BLAS.

name	options	dim	b-width	scalar	matrix	x-vector	scalar	y-vector
_GEMV(	TRANS,	M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_GBMV(	TRANS,	M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_HEMV(	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_HBMV(	UPLO,	N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_HPMV(	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)
_SYMV(	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_SBMV(	UPLO,	N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)
_SPMV(	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)
_TRMV(	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)		
_TBMV(	UPLO, TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)		
_TPMV(	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)		
_TRSV(	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)		
_TBSV(	UPLO, TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)		
_TPSV(	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)		

name	options	dim	scalar	x-vector	y-vector	matrix
GER(		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)
_HER (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)
_HPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)
_HER2(	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)
_HPR2(	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)
_SYR (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)
_SPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)
_SYR2(	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)
_SPR2(	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)