A Proposal for 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 Math 2646
Sandia National Laboratory
Albuquerque, New Mexico 87185

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

Part 1: The Proposal

1. Introduction

In 1973 Hanson, Krogh, and Lawson wrote an article in the SIGNUM Newsletter (Vol. 8, no. 4, page 16) describing 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 [7,8], 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*

†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.

standard for the elementary vector operations.

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 and 3]); and 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 propose the specifications of 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.

We encourage readers interested in such a standardization effort to contact us with their thoughts on the subject. Indeed we wish to solicit alternative ideas and encourage discussion.

In this paper we shall refer to the existing BLAS of Lawson et al. as "Level 1 BLAS" or "Existing BLAS", and the proposed new set as "Level 2 BLAS" or "Extended BLAS". The Level 2 BLAS involve $O(mn)$ scalar operations where m and n are the dimensions of the vector 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.

We plan to make available a complete set of Level 2 BLAS in Fortran 77 so that software developers without access to specific implementations can make use of them. We also plan to develop a test program so that implementations of the extended BLAS can be thoroughly tested before being distributed. We intend eventually to submit the test program and the Fortran 77 version of the routines for publication as an ACM algorithm.

2. Scope of the Extended BLAS

We propose that the following three types of basic operation be performed by the extended BLAS:

a) Matrix-vector products of the form

$y \leftarrow \alpha Ax + y$, $y \leftarrow \alpha A^T x + y$, and $y \leftarrow \alpha A^H x + y$

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

$x \leftarrow Tx$, $x \leftarrow T^T x$, and $x \leftarrow T^H x$,

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

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

$A \leftarrow \alpha xy^T + A$, $A \leftarrow \alpha xy^H + A$, $H \leftarrow \alpha xx^H + H$, and $H \leftarrow \alpha xy^H + \bar{\alpha} yx^H + H$,

where H is a Hermitian matrix.

c) Solution of triangular equations of the form

$x \leftarrow T^{-1}x$, $x \leftarrow T^{-T}x$, and $x \leftarrow T^{-H}x$,

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

We propose that, where appropriate, the operations be 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.

See Part 2 of this report for examples to illustrate the uses and advantages of the proposed routines, and an example to illustrate the implementation of the routines is given in Appendix A.

3. Naming Conventions

The proposed name of a Level 2 BLAS is in the LINPACK style and consists of five characters (except for four routine names with six characters). The fourth and fifth characters in the name denote the type of operation, as follows:

MV - Matrix-vector product
R1 - Rank-one update
R2 - Rank-two update
IV - Inverse matrix-vector product
(i.e., solution of a set of linear equations)

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 in packed form
TB - Triangular band matrix

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)

The proposed available combinations are indicated in the table 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 proposed collection of routines can be thought of as being divided into four separate parts, *complex*, *real*, *double precision*, and *complex*16.*. Each part will include a separate testing program. The routines proposed are written in the ANSI Fortran 77 standard with the exception of the routines that use COMPLEX*16 variables. These routines are included for completeness and, because the Fortran standard does not provide for this variable type, may not be available on all machines.

Table 3.1

complex	*real*	MV	R1	R2	IV
CGE	SGE	*	*†		
CGB	SGB	*			
CHE	SSY	*	*	*	
CHP	SSP	*	*	*	
CHB	SSB	*			
CTR	STR	*			*
CTP	STP	*			*
CTB	STB	*			*

We do not propose 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 B.

4. Parameter Conventions

We propose a similar convention for the parameter lists to that for the existing BLAS, but with extensions where comparable parameters are not present in the existing BLAS. The proposed order of parameters is as follows:

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

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

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

† For the general rank-1 update (GER1) we propose two complex routines: CGER1C for $A \leftarrow \alpha xy^H + A$ and CGER1U for $A \leftarrow \alpha xy^T + A$. This is the only exception to the one to one correspondence between real and complex routines and the only exception to the five-character naming conventions. See section 7 for further discussion.

Value	Meaning
' ' or '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.

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

The size of the matrix is determined by the two parameters M and N for an m by n rectangular matrix; by the parameters M, N, KL, and KU for an m by n band matrix with kl sub-diagonals and ku super-diagonals; and by the parameters N and K for an n by n symmetric or Hermitian or triangular band matrix with k super- and/or sub-diagonals.

The description of the matrix consists either of the array name (A) followed by the leading dimension of the array as declared in the calling (sub) program (LDA), when the matrix is being stored in a two- dimensional array; or the Fortran array name (AP) alone when the matrix is being stored as a (packed) vector. When A is not band, then in the former case the actual array must contain at least $(m + d(n-1))$ elements, where d is the leading dimension of the array and $m = n$ for a square matrix; and in the latter case the actual array must contain at least $n(n+1)/2$ elements.

The scalar always has the dummy argument name ALPHA. 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 negative, zero, or positive. When the vector x consists of k elements, then the corresponding actual array argument X must be of length at least $(1 + (k-1)|INCX|)$.

The final parameter of each routine is an error indicator INFO which is set to:

0	if the operation is successfully completed.
i > 0	if the i^{th} parameter has an illegal value on entry.
-i < 0	if an attempt is made to compute $x \leftarrow T^{-1}x$, $x \leftarrow T^{-T}x$, or $x \leftarrow T^{-H}x$, and the i^{th} diagonal element of T is exactly zero.

The following values of parameters are assumed to be illegal:

Any value of the character parameters DIAG, TRANS, or UPLO whose meaning is not specified.
M < 0 for GE and GB routines.
N < 0 for all routines.
KL < 0 or KL < M - N or KL > M - 1 for the GB routines
KU < 0 or KU < N - M or KU > N - 1 for the GB routines
K < 0 or K > N - 1 for the HB, SB, and TB routines.
LDA < 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.

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.

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 to be stored in either 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 column 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 parameter UPLO serves to define whether the matrix is upper (UPLO = 'U') or lower (UPLO = 'L') triangular. In the 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 leading dimension of the matrix is stored in the $ku+1^{th}$ row 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 the $k+1^{th}$ row 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 the $k+1^{th}$ row 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 R1 and R2 routines these imaginary parts will be set to zero on return.

For packed triangular matrices the same storage layout is used whether or not DIAG = 'U' (diagonal elements are assumed to have the value 1), i.e. space is left for the diagonal elements even if those array elements are not referenced.

6. Specification of the Extended BLAS

(This section has been deleted for space reasons. Consult the full proposal for details.)

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.

The parameter α is included in the non-triangular routines to give extra flexibility, but we recommend that implementors consider special code for the cases where $\alpha = 1.0$ and $\alpha = -1.0$. Similarly, as with the level 1 BLAS, we have included an increment parameter with the vectors so that a vector could, for example, be a row of a matrix. But again we recommend that implementors consider special code for the case where the increments are unity.

As noted earlier, corresponding to the real routine SGER1 we propose two complex routines CGER1C (for $A \leftarrow \alpha x y^H + A$) and CGER1U (for $A \leftarrow \alpha x y^T + A$). Both are frequently required. An alternative would be to provide a single complex routine CGER1 with an option parameter; however this parameter would have become redundant in the real routine SGER1. Rather than have redundant parameters, or different parameter lists for the real and complex routines, we have chosen two distinct complex routines; they are analogous to the level 1 BLAS CDOTC ($c \leftarrow c + x^H y$) and CDOTU ($c \leftarrow c + x^T y$).

We have included an error parameter INFO in each of the routines. This is a departure from the conventions of the level 1 BLAS, but is prompted by the increased possibilities for error (incorrect dimensioning of 2-dimensional arrays, division by zero in the solution of triangular sets of equations), and the decrease in the relative cost of checking for errors.

The CHARACTER parameters UPLO, TRANS, DIAG, are to be comprised of text in the Fortran character set. This convention will be adhered to in the testing programs and the portable Fortran version of the Level 2 BLAS. 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 parameter 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 characters 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 develop

special versions of the Extended BLAS package that 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 [6,9] 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 $y \leftarrow \alpha Ax + y$ 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.

Although not discussed here, we plan to provide a portable testing package for each of the four parts of this extended BLAS package. The test package will be self-contained; generating test data and checking for conformity with this proposal in a portable fashion.

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. 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.

In this section we have tried to explain some of the design decisions which we have taken. We welcome comment from people who feel that we have overlooked important considerations, or at least have not attached enough weight to them.

Part 2:

8. Applicability of a Set of Level 2 BLAS

The purpose of Part 2 is to demonstrate the wide applicability of the set of Level 2 BLAS proposed in Part 1. LINPACK and EISPACK are taken as well-known examples of heavily used software which could, with hindsight, have made extensive use of such a set of BLAS; future versions may do just that. The most straightforward way to use the Level 2 BLAS is as components in the development of new software; working out how to fit them into existing software can be more complicated, but may still be very worthwhile if it leads to substantial improvements in performance; a similar exercise has already been undertaken on selected NAG Library routines to improve their performance on vector processors [1].

Sections 10 and 11 demonstrate in some detail how calls to individual Level 2 BLAS can be used to perform the bulk of the computation in the CPO-, CPP- and CPB-sets of LINPACK routines. Section 12 surveys the broad applicability of the entire set of Level 2 BLAS in LINPACK and EISPACK.

9. Example: the CPO- Routines in LINPACK

(This section has been deleted for space reasons. Consult the full proposal for details.)

10. Extensions: the CPP- and CPB- Routines in LINPACK

(This section has been deleted for space reasons. Consult the full proposal for details.)

11. Scope of Application in LINPACK and EISPACK

Table 1 shows, without any claim to completeness, where calls to the proposed set of Level 2 BLAS might be incorporated into LINPACK or EISPACK routines (only the real single precision set have been considered, and EISPACK routines which operate on complex matrices have been omitted). Many of the calls to SGEMV and SGER1 arise in the application of a Householder transformation to a matrix. The computation

$$X \leftarrow (I - \alpha u u^T)X = X - \alpha u (X^T u)^T$$

can be performed by the following operations using the Level 2 BLAS in conjunction with a work vector w:

$$w \leftarrow 0$$
$$w \leftarrow X^T u + w \qquad (SGEMV)$$
$$X \leftarrow -\alpha u w^T + X \qquad (SGER1)$$

Some latitude has been allowed in drawing up Table 1, in that a suitable work vector may not be available in the existing code. Similarly, to make use of STRIV in SGESL would require the interchanges to be handled differently by the SGE- set of LINPACK routines.

Furthermore, many of the algorithms listed in Table 1 can be organized in different ways, as was pointed out by Dongarra, Gustavson and Karp [5] for matrix multiplication and LU-factorization. Following their convention, in which the basic operation in the innermost loop is something like $a_{ij} \leftarrow a_{ij} \pm a_{ik} a_{kj}$:

- with i as the outermost loop index, the result is computed row by row;

- with j as the outermost loop index, the result is computed column by column;

- with k as the outermost loop index, the computation proceeds by updating a large submatrix and (in most algorithms) reducing the dimension of the problem by one at each stage.

Table 1

BLAS	LINPACK	EISPACK	BLAS	LINPACK	EISPACK
SGEMV	SGEDI	ORTHES	SSPMV	-	TRED3
	SQRDC	ORTBAK			
	SSVDC	ORTRAN	SSPR1	SPPDI	-
		ELMHES		SSPFA	
		QZHES			
		TRED2	SSPR2	-	TRED3
		TRBAK1			
		TRBAK3	STRMV	-	ELMBAK
		REDUC			
			STPMV	SPPDI	-
SGER1	SPODI	ORTHES			
	SGEFA	ORTBAK	STRIV	SGESL	ELMHES
	SGEDI	ORTRAN		SPOFA	REDUC
	SGBFA	QZHES		SPBFA	REBAK
	SQRDC	TRED2		SPOSL	
	SSVDC	TRBAK1			
		TRBAK3	STPIV	SPPSL	-
				SPPFA	
SSYMV	-	TRED1			
		TRED2	STBIV	SGBSL	-
				SPBSL	
SSYR1	SCHDC	-			
	SPODI				
	SSIFA				
SSYR2	-	TRED1			
		TRED2			

In Fortran code, in order to avoid paging problems, the j-form is usually preferred to the i-form, but the i-form would be preferred if matrices were stored by rows. The i-form and j-form can often permit GAXPY operations [5] in their innermost loops, so are likely to be favorable on a CRAY-1, whereas the k-form (which can work either by row or column) would be preferable on machines which allowed the individual AXPY operations of a rank-1 update to be performed in parallel. Each of the

three methods of organizing an algorithm can involve calls to different Level 2 BLAS. For example, for LU-factorization,

- the i-form requires STRIV ($x \leftarrow U^{-T}x$) and SGEMV ($y \leftarrow \alpha A^T x + y$)

- the j-form requires STRIV ($x \leftarrow L^{-1}x$) and SGEMV ($y \leftarrow \alpha Ax + y$)

- the k-form requires SGER1 ($A \leftarrow \alpha xy^T + A$)

In all the algorithms studied which involve symmetric matrices, it has proved possible to find a form which permits the use of packed storage in conjunction with the Level 2 BLAS. Consider as an example the computation of $L^{-1}AL^{-T}$ for symmetric A using packed storage (i.e. a "packed" version of the EISPACK routine REDUC). If the lower triangles of A and L are packed by columns, the computation can be implemented by calls to SSPR2 ($A \leftarrow \alpha xy^T + \alpha yx^T + A$) and STPIV ($x \leftarrow L^{-1}x$); if the lower triangles are packed by rows, this is equivalent to computing $U^{-T}AU^{-1}$ with the upper triangles of A and U packed by columns, and can be implemented by calls to STPIV ($x \leftarrow U^{-1}x$) and SSPMV ($y \leftarrow \alpha Ax + y$ for symmetric A).

A further degree of variation is introduced if for any reason we wish to perform the basic triangular factorizations 'backwards' (i.e. as UL, L^TL or UU^T).

Allowing for all these possibilities within a small set of fundamental algorithms, yields applications for each of the proposed Level 2 BLAS, except the banded matrix vector products. We consider it important that the set of extended BLAS be sufficiently wide to permit this degree of variation and not to constrain the freedom of software developers.

These remarks have concentrated on algorithms for dense matrices with a simple structure. It is hoped that the Level 2 BLAS may also prove useful when dealing with matrices with more complicated structures, if this can be done by splitting them into dense sub-matrices.

12. Acknowledgements

An earlier draft of this proposal was discussed at the Parvec IV Workshop organized by John Rice at Purdue University on October 29-30, 1984. We wish to thank all the participants at that workshop for their comments, discussions, and encouragement.

13. References

[1] C. Daly and J.J. Du Croz. "Performance of a Subroutine Library on Vector- Processing Machines". *To be published in Computer Physics Communications.*

[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, F.G. Gustavson, and A. Karp. "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine". *SIAM Review*, 26, 91-112 (1984).

[6] 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.

[7] 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.

[8] 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.

[9] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, Vol. 6, 2nd edition, Springer-Verlag, Berlin, 1976.

Appendix A

(This section has been deleted for space reasons. Consult the full proposal for details.)

Appendix B

(This section has been deleted for space reasons. Consult the full proposal for details.)

Appendix C

This appendix contains the calling sequences for all the proposed level 2 BLAS.

```
 name      options          dim   b-width scalar matrix  x-vector y-vector info

_GEMV(         TRANS,       M, N,          ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_GBMV(         TRANS,       M, N, KL, KU, ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_HEMV( UPLO,                   N,          ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_HBMV( UPLO,                   N,     K,   ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_HPMV( UPLO,                   N,          ALPHA, AP,     X, INCX, Y, INCY, INFO )
_SYMV( UPLO,                   N,          ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_SBMV( UPLO,                   N,     K,   ALPHA, A, LDA, X, INCX, Y, INCY, INFO )
_SPMV( UPLO,                   N,          ALPHA, AP,     X, INCX, Y, INCY, INFO )
_TRMV( UPLO, TRANS, DIAG,      N,                 A, LDA, X, INCX,          INFO )
_TBMV( UPLO, TRANS, DIAG,      N,     K,          A, LDA, X, INCX,          INFO )
_TPMV( UPLO, TRANS, DIAG,      N,                 AP,     X, INCX,          INFO )

_TRIV( UPLO, TRANS, DIAG,      N,                 A, LDA, X, INCX,          INFO )
_TBIV( UPLO, TRANS, DIAG,      N,     K,          A, LDA, X, INCX,          INFO )
_TPIV( UPLO, TRANS, DIAG,      N,                 AP,     X, INCX,          INFO )

 name      options          dim   scalar x-vector y-vector matrix  info

_GER1_(                    M, N, ALPHA, X, INCX, Y, INCY, A  LDA, INFO )
_HER1(  UPLO,                 N, ALPHA, X, INCX,          A, LDA, INFO )
_HPR1(  UPLO,                 N, ALPHA, X, INCX,          AP,     INFO )
_HER2(  UPLO,                 N, ALPHA, X, INCX, Y, INCY, A, LDA, INFO )
_HPR2(  UPLO,                 N, ALPHA, X, INCX, Y, INCY, AP,     INFO )
_SYR1(  UPLO,                 N, ALPHA, X, INCX,          A, LDA, INFO )
_SPR1(  UPLO,                 N, ALPHA, X, INCX,          AP,     INFO )
```

```
_SYR2(  UPLO,                N, ALPHA, X, INCX, Y, INCY, A, LDA, INFO )
_SPR2(  UPLO,                N, ALPHA, X, INCX, Y, INCY, AP,     INFO )
```