/

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-660

A Proposal for Standard Linear Algebra Subprograms

R. J. Hanson
Washington State University

F. T. Krogh and C. L. Lawson
Jet Propulsion Laboratory

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

November 15, 1973

Prepared Under Contract No. NAS 7-100
National Aeronautics and Space Administration

i

PREFACE

The work described in this report was performed by the Data Systems Division of the Jet Propulsion Laboratory.

Correspondence should be directed to either R. J. Hanson, Dept. of Computer Science, Johnson Hall, Washington State University, Pullman, Washington, 99163, or F. T. Krogh and C. L. Lawson, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, California, 91103.

CONTENTS

PRECEDING PAGE BLANK NOT FILMED

ABSTRACT

This memorandum proposes a set of Fortran callable subprograms which will be useful in the development of efficient portable ANSI Fortran subprograms and applications programs in the area of linear algebra.

Introduction

The purpose of this report is to propose a set of standard subprograms (modules) for performing many of the elementary operations of numerical linear algebra. The goal is to make it more feasible to produce efficient portable Fortran programs in the area of linear algebra.

By adopting a set of standard subprogram names and parameter lists for certain fundamental operations it becomes worthwhile to make the effort of producing efficient assembly coded implementations of these subprograms for a wide variety of different computer systems. For example, it has been found [Krogh (1)] that the use of assembly coded modules in a double precision program for solving linear equations based on Householder transformations with column scaling and column interchanges reduced the execution time on a Univac 1108 by 15% to 30% relative to the time required when carefully written Fortran modules were used.

We intend this report as a specific proposal about which discussion of these ideas can be focused. We hope that readers of this report will communicate to us their thoughts on the general usefulness of such a set of standard subprograms as well as comments on specific details.

We separate this proposed set of subprograms into two classes to emphasize the different levels of importance which we attach to the two classes.

Class I contains subprograms implementing the operations that occur as the most frequently executed innermost loops in many of the fundamental algorithms of linear algebra. For this reason increases in efficiency of executing these operations may be expected to be particularly significant in improving the efficiency of the programs in which they are used.

We feel that it would represent a significant advance in the technology of supporting efficient portable Fortran programs if carefully programmed assembly coded subprograms were available on a wide variety of computer systems for these Class I functions and if persons writing portable Fortran subprograms for the fundamental algorithms of linear algebra would use these Class I functions where applicable.

The operations which we feel belong in Class I according to the above stated criteria are: (1) the dot product (inner product) of two vectors, (2) the elementary vector operation, $y := ax + y$ where x and y are n-vectors and a is a scalar, and (3) the Givens 2 x 2 orthogonal transformation applied to a 2 x n submatrix.

In the case of the Givens transformation there are presently two competitive forms in which it can be implemented, the standard (4-multiply, 2-add) form and a modified (2-multiply, 2-add, no square root) form. For the sake of discussion we give proposed subprogram specifications for both of these forms.

Note that we have omitted from Class I versions of the dot product which require arithmetic precision extended beyond that of standard double-precision or the notion of double-precision complex arithmetic since these concepts are not at present defined in ANSI FORTRAN. There is a chance that when the revised Fortran standard [FORTREV] is finalized by ANSI it will define a double-precision complex variable type. We propose that the specification of subprograms using double-precision complex arithmetic should await finalization of FORTREV.

The subprograms in Class II represent operations which occur frequently in computational linear algebra but not in the most frequently executed loops. Thus their efficiency may have less impact than Class I subprograms. In fact however it is not easy to predict the relative importance of different subprograms in a particular application and actual timings may show Class II subprograms to be of more importance than one might expect. For instance in the tests mentioned previously, [Krogh (1)], about 1/3 of the saving at N = 100 and about 1/2 at N = 15 were due to the assembly coded module for the euclidean norm while the rest of the savings were due to the assembly coded modules for the inner product and the elementary vector operation.

Aside from the question of efficiency the Class II subprograms are proposed for their potential convenience in the writing of programs for linear algebraic problems. The use of these subprograms may improve the self-documenting quality of the Fortran programs in which they are used and may tend to reduce the occurrence of coding errors by relieving the user of one level of coding detail.

For the purpose of experimenting with this approach to writing linear algebra programs we have produced Fortran coded and Univac 1108 assembly language coded subprograms for most of the Class I modules and Fortran coded subprograms for most of the Class II modules.

After allowing a period of time for discussion of these ideas it is our plan to bring the following set of subprograms into being:

(1) Portable Fortran coded subprograms for all of the modules except for the extended precision inner-product module.

(2) Assembly coded subprograms for Class I modules for as many different machines as possible.

We invite persons willing to undertake the assembly coding of these Class I modules on machines other than the Univac 1108 and the IBM 360/75 to do so and communicate the resulting code to us. We will depend upon such contributions to achieve wide machine coverage with this set of subprograms.

Persons willing to write assembly versions of these subprograms for different machines may obtain listings of Fortran versions and test drivers from the authors.

Convention Regarding Vector Storage

An N-vector $x = (x_1, \ldots, x_N)$ to be referenced by any subprogram described in this memorandum will be identified in the subprogram parameter list by a pair of symbols, say X and INCX, where X is the name of an array of type REAL, DOUBLE PRECISION, or COMPLEX, as appropriate, and INCX is an INTEGER type variable indicating the index increment between successive components of x in the array X. The location of components of the vector x within the array X is specified as follows:

If INCX $\geq$ 0	$X(1 + (I - 1)*INCX) = x_I$	$I = 1, \ldots, N$
If INCX < 0	$X(1 + (N - I)* \lvert INCX \rvert) = x_I$	$I = 1, \ldots, N$

For example if N = 4 and INCX = 2 we have X(1) = x_1, X(3) = x_2, X(5) = x_3, and X(7) = x_4, while if N = 4 and INCX = -2 we have X(7) = x_1, X(5) = x_2, X(3) = x_3, and X(1) = x_4.

In some of the subprograms, as noted below, INCX is required to be nonnegative.

Convention Regarding $N \leq 0$

FUNCTION subprograms return the value zero when $N \leq 0$. Exceptions are those FUNCTIONs which return an index. These return the index, 1, if $N \leq 0$.

Proposed Subprograms in Class I

Operation	Usage Statement	Remarks
$w := \sum_{i=1}^{N} x_i y_i$	SW = SDOT(N, SX, INCX, SY, INCY)	SW, SDOT, SX( ), SY( ), single precision
$w := \sum_{i=1}^{N} x_i y_i$	DW = DSDOT(N, SX, INCX, SY, INCY)	DW, DSDOT double precision SX( ), SY( ) single precision
$w := \sum_{i=1}^{N} x_i y_i$	DW = DDOT(N, DX, INCX, DY, INCY)	DW, DDOT, DX( ), DY( ) double precision
$w := \sum_{i=1}^{N} x_i y_i$	CW = CDOT(N, CX, INCX, CY, INCY, 0)	CW, CDOT, CX( ), CY( ) complex
$w := \sum_{i=1}^{N} \bar{x}_i y_i$	CW = CDOT(N, CX, INCX, CY, INCY, 1)	CW, CDOT, CX( ), CY( ) complex
$y := ax+y$	CALL SELVOP(N, SA, SX, INCX, SY, INCY)	SA, SX( ), SY( ) single precision.
$y := ax+y$	CALL DELVOP(N, DA, DX, INCX, DY, INCY)	DA, DX( ), DY( ) double precision.
$y := ax+y$	CALL CELVOP(N, CA, CX, INCX, CY, INCY)	CA, CX( ), CY( ) complex.
Apply Givens reflection matrix $\begin{bmatrix} c & s \\ s & -c \end{bmatrix}$ to the 2 x n matrix $\begin{bmatrix} x_1 \cdots x_n \\ y_1 \cdots y_n \end{bmatrix}$	CALL SG2(N, SX, INCX, SY, INCY, C, S)	SX( ), SY( ), C, and S single precision See SG1 in Class II for computation of C and S.
Apply Givens reflection matrix $\begin{bmatrix} c & s \\ s & -c \end{bmatrix}$ to the 2 x n matrix $\begin{bmatrix} x_1 \cdots x_n \\ y_1 \cdots y_n \end{bmatrix}$	CALL DG2(N, DX, INCX, DY, INCY, DC, DS)	DX( ), DY( ), DC and DS are double precision. See DG1 in Class II for computation of DC and DS.
Apply the modified Givens reflection transformation [See Appendix A] to the 2 x n matrix $\begin{bmatrix} x_1 \cdots x_n \\ y_1 \cdots y_n \end{bmatrix}$	CALL SMG2(N, SX, INCX, SY, INCY) [uses COMMON/SMG3/IFLAG1, IFLAG2, T1, T2, SRA, SRB]	SX( ), SY( ) T1, T2, SRA, and SRB single precision. See SMG1 in Class II for computation of the quantities in COMMON.
Apply the modified Givens reflection transformation [See Appendix A] to the 2 x n matrix $\begin{bmatrix} x_1 \cdots x_n \\ y_1 \cdots y_n \end{bmatrix}$	CALL DMG2(N, DX, INCX, DY, INCY) [uses COMMON/DMG3/IFLAG1, IFLAG2, DT1, DT2, DRA, DRB]	DX( ), DY( ), DT1, DT2, DRA, DRB double precision. See DMG1 in Class II for computation of the quantities in COMMON.

Proposed Subprograms in Class II

Operation	Usage Statement	Remarks
Compute $r := (a^2+b^2)^{1/2}$ $c := a/r$, $s := b/r$ $a := r$ defining the Givens reflection matrix $\begin{bmatrix} c & s \\ s & -c \end{bmatrix}$	CALL SG1(A, B, C, S)	A, B, C, and S single precision
Compute $r := (a^2+b^2)^{1/2}$ $c := a/r$, $s := b/r$ $a := r$ defining the Givens reflection matrix $\begin{bmatrix} c & s \\ s & -c \end{bmatrix}$	CALL DG1(DA, DB, DC, DS)	DA, DB, DC, and DS double precision.
Compute parameters defining a modified Givens reflection. See Appendix A.	CALL SMG1(SCA, SCB, A, B) [uses COMMON/ SMG3/IFLAG1, IFLAG2, T1, T2, SRA, SRB]	SCA, SCB, A, B, T1, T2, SRA, SRB single precision; IFLAG1, IFLAG2 integer.
Compute parameters defining a modified Givens reflection. See Appendix A.	CALL DMG1(DSCA, DSCB, DA, DB) [uses COMMON/ DMG3/IFLAG1, IFLAG2, DT1, DT2, DRA, DRD]	DSCA, DSCB, DA, DB, DT1, DT2, DRA, DRB double precision; IFLAG1, IFLAG2 integer.
$w := b + \sum_{i=1}^{N} x_i y_i$	DW = DXDOT(N, DX, INCX, DY, INCY, DB, XC, 0)	DW, DXDOT, DX( ), DY( ), DB, double precision; XC extended precision; XC is represented with a single precision array of length 5; XC is replaced by $b + \sum_{i=1}^{N} x_i y_i$.
$w := b + \sum_{i=1}^{N} x_i y_i + c$	DW = DXDOT(N, DX, INCX, DY, INCY, DB, XC, 1)	DW, DXDOT, DX( ), DY( ), DB double precision; XC extended precision; XC is replaced by $b + \sum_{i=1}^{N} x_i y_i + c$.
Copy x into y	CALL SCOPY(N, SX, INCX, SY, INCY)	SX( ), SY( ) single precsion
Copy x into y	CALL DCOPY(N, DX, INCX, DY, INCY)	DX( ), DY( ) double precision
Copy x into y	CALL CCOPY(N, CX, INCX, CY, INCY)	CX( ), CY( ) complex
Interchange x and y	CALL SSWAP(N, SX, INCX, SY, INCY)	SX( ), SY( ) single precision
Interchange x and y	CALL DSWAP(N, DX, INCX, DY, INCY)	DX( ), DY( ) double precision
Interchange x and y	CALL CSWAP(N, CX, INCX, CY, INCY)	CX( ), CY( ) complex

In all of the remaining subprograms INCX ≥ 0 is required.

$w := \left[\sum_{i=1}^{N} x_i^2\right]^{1/2}$	SW = S2NRM(N, SX, INCX)	SW, S2NRM, SX( ) single precision
$w := \left[\sum_{i=1}^{N} x_i^2\right]^{1/2}$	DW = D2NRM(N, DX, INCX)	DW, D2NRM, DX( ) double precision

Proposed Subprograms in Class II (contd)

Operation	Usage Statement	Remarks
$w := \left[\sum_{i=1}^{N} \lvert x_i\rvert^2\right]^{1/2}$	SW = SC2NRM(N, CX, INCX)	SW, SC2NRM, single precision; CX( ) complex.
$w := \left[\sum_{i=1}^{N} \lvert x_i\rvert^2\right]^{1/2}$	SW = SD2NRM(N, SX, INCX)	SW, SD2NRM, SX( ) single precision. Double precision arithmetic used internally.
$w := \left[\sum_{i=1}^{N} \lvert x_i\rvert^2\right]^{1/2}$	DW = DX2NRM(N, DX, INCX)	DW, DX2NRM, DX( ) double precision. Extended precision arithmetic used internally.
$w := \sum_{i=1}^{N} \lvert x_i\rvert$	SW = SASUM(N, SX, INCX)	SW, SASUM, SX( ) single precision
$w := \sum_{i=1}^{N} \lvert x_i\rvert$	DW = DASUM(N, DX, INCX)	DW, DASUM, DX( ) double precision
$w := \sum_{i=1}^{N} \{\lvert \mathrm{Real}(x_i)\rvert + \lvert \mathrm{Imag}(x_i)\rvert\}$	SW = SCASUM(N, CX, INCX)	SW, SCASUM single precision; CX( ) complex.
$x := ax$	CALL SSCALE(N, SA, SX, INCX)	SA, SX( ) single precision
$x := ax$	CALL DSCALE(N, DA, DX, INCX)	DA, DX( ) double precision
$x := ax$	CALL CSCALE(N, CA, CX, INCX)	CA, CX( ) complex
$x := ax$	CALL CSSCAL(N, SA, CX, INCX)	SA single precision; CX( ) complex.
Find indices i_{min} and i_{max} corresponding to the smallest and largest components of x.	CALL SMNMX(N, SX, INCX, IMIN, IMAX)	SX( ) single precision
Find indices i_{min} and i_{max} corresponding to the smallest and largest components of x.	CALL DMNMX(N, DX, INCX, IMIN, IMAX)	DX( ) double precision
Find an index i_{max} corresponding to the maximum absolute value of components of the vector x.	IMAX = ISAMAX(N, SX, INCX)	SX( ) single precision
Find an index i_{max} corresponding to the maximum absolute value of components of the vector x.	IMAX = IDAMAX(N, DX, INCX)	DX( ) double precision
Find an index i_{max} identifying the component of the complex vector x having maximum sum of magnitudes of real and imaginary parts.	IMAX = ICAMAX(N, CX, INCX)	CX( ) complex

APPENDIX A

A MODIFIED GIVENS TRANSFORMATION

Introduction

The standard Givens reflection transformation is a 2 x 2 matrix $G \equiv \begin{bmatrix} c & s \\ s & -c \end{bmatrix}$. For an arbitrary 2 vector $x = \begin{bmatrix} a \\ b \end{bmatrix}$, the entries c and s are computed by $r = (a^2 + b^2)^{1/2}$, $c = a/r$ and $s = b/r$, and this gives $Gx = \begin{bmatrix} r \\ 0 \end{bmatrix}$.

The modified Givens transformation [Gentleman (2) and (3)] is a variation of the standard Givens transformation. This is done by considering the vector x written in factored form $x = D_1 x_1$. Here D_1 is a 2 x 2 diagonal matrix. The matrix G is constructed so that $Gx = \begin{bmatrix} r \\ 0 \end{bmatrix}$. The identity $GD_1 \equiv D_2 H$ is the critical point. (Here the matrix D_2 is 2 x 2 and diagonal.) The 2 x 2 matrix H has one of the two forms.

$$H = \begin{bmatrix} t_1 & 1 \\ 1 & -t_2 \end{bmatrix} \quad \text{or} \quad H = \begin{bmatrix} 1 & t_1 \\ t_2 & -1 \end{bmatrix}$$

With the standard Givens transformation formation of the product matrix $GA_{2 \times n}$ requires 4n multiplications and 2n additions. With the modified Givens transformation the matrix A is always considered in factored form $A = D_1 A_1$. Noting that $GA = G(D_1 A_1) = D_2(HA_1)$, allows the matrix $A_2 \equiv HA_1$ to be computed with 2n multiplications and 2n additions because of the way the matrix H is defined above.

No Square Roots

Only squares of elements of the 2 x 2 diagonal matrix D_1 are involved in the computation of the matrix H and (the squares of) the elements of the updated 2 x 2 diagonal matrix D_2. This remark obviates the need for computing square roots.

Rescaling

The elements of the diagonal matrices generated will generally decrease as further transformations are constructed. This may require occasional rescaling to avoid underflow danger. The identity $D_1 A_1 = (D_1 S^{-1})(SA_1) \equiv \tilde{D}_1 \tilde{A}_1$ (S an arbitrary 2 x 2 diagonal matrix) shows that the factorization can be rescaled whenever desired.

We have provided for rescaling to be performed whenever either diagonal term of the matrix D_2 is less than 2^{-12}.

The matrix D_1 can be initialized to be the 2 x 2 identity matrix, for example.

Coding Details

The modified Givens transformation is called with the Fortran statement

```
CALL SMG1(SCA, SCB, A, B)
```

The standard Givens transformation which is implicitly constructed in this subroutine eliminates the second entry of the 2-vector

$$\begin{bmatrix} SCA^{1/2} * A \\ SCB^{1/2} * B \end{bmatrix}$$

The first entry of the product matrix $H\begin{bmatrix} A \\ B \end{bmatrix}$ replaces A in storage. The squares of the entries of the diagonal matrix D_2 replace SCA and SCB in storage.

When rescaling occurs both the first entry of $H\begin{bmatrix} A \\ B \end{bmatrix}$ and the diagonal terms SCA and SCB are appropriately modified.

The transformation is applied to a 2 x n matrix $D_1 \begin{bmatrix} x_1 \ldots x_n \\ y_1 \ldots y_n \end{bmatrix}$ with the Fortran statement.

```
CALL SMG2(N, X, INCX, Y, INCY).
```

The various parameters necessary to apply this transformation are communicated from SMG1 to SMG2 by means of the COMMON statement

```
COMMON /SMG3/IFLAG1, IFLAG2, T1, T2, SRA, SRB
```

These parameters are defined as follows:

IFLAG1	Indicates the form of the matrix H: IFLAG1 = 0 when $H = \begin{bmatrix} t_1 & 1 \\ 1 & -t_2 \end{bmatrix}$ IFLAG1 = 1 when $H = \begin{bmatrix} 1 & t_1 \\ t_2 & -1 \end{bmatrix}$
IFLAG2	Rescaling indicator IFLAG2 = 0 if no rescaling is required IFLAG2 = 1 when rescaling is required
T1, T2	Nonunit entries of the appropriate form of the 2 x 2 matrix H.
SRA, SRB	Diagonal entries of the diagonal matrix S which will be applied to the matrix $H \begin{bmatrix} x_1 \dots x_n \\ y_1 \dots y_n \end{bmatrix}$ when IFLAG2 = 1. The values of SRA and SRB, when IFLAG2 = 1, are either 2^{24}, 2^{12}, or 1.

APPENDIX B

SOME IMPLEMENTATIONS OF DSDOT

To provide concrete examples of our present thinking on implementations of these subprograms we include listings of three versions of DSDOT. Figure B-1 is Fortran code, Figure B-2 is Univac 1108 assembly code and Figure B-3 is IBM 360/75 assembly code.

In the Fortran code note the use of DBLE to achieve double length accumulation. With some Fortran compilers double length accumulation would occur even without the DBLE just because DSDOT is of type DOUBLE PRECISION. The DBLE is necessary however on some compilers. This leads to unnecessary inefficiency on some compilers which would not need the DBLE. This is an example of the present impossibility of writing portable Fortran code which will be efficient on a variety of machines.

Another problem arises in handling indexing with the index spacing specified by the parameters INCX and INCY. The statement at line 19 generates quite efficient machine code on the Univac 1108 but is not within the specifications of ANS Fortran. The alternative code given as comments in lines 21-23 is legal ANS Fortran but generates much less efficient code on the Univac 1108.

The comment cards "C1", "C2", and "C3" delimit the two different versions of Fortran code. The comment cards "C4" and "C5" delimit the initial descriptive comments. We are using these comment cards "C1" through "C5" as reference marks for performing certain editing operations on the whole set of subprograms.

```
      DOUBLE PRECISION FUNCTION DSDOT(N,X,INCX,Y,INCY)
C4
C     DOT PRODUCT OF SNGL. PREC. VECTORS USING DBLE. PREC. ACCUMULATION.
C
C     COMPUTES DSDOT = SUM FROM 1 TO N OF A(I)*B(I) WHERE
C     A(I) = X(1-INCX+I*INCX)     IF INCX .GE. 0
C     A(I) = X(1-N*INCX+I*INCX)     IF INCX .LT. 0
C     B(I)   DEFINED SIMILARLY WITH X AND INCX REPLACED BY Y AND INCY
C5
      REAL X(1),Y(1)
      DSDOT = 0.D0
      IF(N .LE. 0)RETURN
      IF(INCX .LT. 0) GO TO 2
      KX = -INCX + 1
      GO TO 4
    2 KX = -N*INCX + 1
    4 IF(INCY .LT. 0) GO TO 6
      KY = -INCY + 1
      GO TO 8
    6 KY = -N*INCY + 1
    8 CONTINUE
          DO 10 I = 1,N
C1
          DSDOT = DSDOT + DBLE(X(I*INCX + KX))*DBLE(Y(I*INCY + KY))
C2
C         JX = I*INCX + KX
C         JY = I*INCY + KY
C         DSDOT = DSDOT + DBLE(X(JX))*DBLE(Y(JY))
C3
   10 CONTINUE
      RETURN
      END
```

Figure B-1. Fortran Code for DSDOT

```
               AXR$
$(1).
.
.    INNER PRODUCT OF SNGL. PREC. VECTORS USING DBLE. PREC. ACCUMULATION.
.
. TO BE USED AS FORTRAN FUNCTION  DSDOT(N,X,INCX,Y,INCY)
. WHERE DSDOT IS OF TYPE DOUBLE PRECISION, X AND Y ARE OF TYPE REAL.
. AND   DSDOT= SUM FROM I=1 TO N OF A(I)*B(I)  WHERE
. A(I) = X(1-INCX+I*INCX)     IF   INCX.GE.0
. A(I) = X(1-N*INCX+I*INCX)   IF   INCX.LT.0
. B(I)    DEFINED SIMILARLY, WITH X AND INCX REPLACED BY Y AND INCY
.
DSDOT*     DSL        A0,72                . STORE 0 IN A0 AND A1
           LR         R3,*0,X11            . STORE N IN R3
           JGD        R3,NPOS              . STORE N-1 IN R3 AND TEST N
           J          6,X11                . EXIT IF N.LE.0
NPOS       DS         A6,SAVE              . SAVE REGISTERS A6 AND A7
           LA,U       A2,*1,X11            . LOAD ADDRESS OF X
           LA,U       A3,*3,X11            . LOAD ADDRESS OF Y
           LXI        A2,*2,X11            . LOAD INCREMENT ON X
           LXI        A3,*4,X11            . LOAD INCREMENT ON Y
           JP         A2,TINCY             . TEST IF INCX.GE.0
           LNA        A4,A2                . ADD -INCX*(N-1)
           SSA        A4,18                .     TO THE BASE
           MSI        A4,R3                .     ADDRESS
           AH         A2,A4                .     FOR X
TINCY      JP         A3,LOOP              . TEST IF INCY.GE.0
           LNA        A4,A3                . ADD -INCY*(N-1)
           SSA        A4,18                .     TO THE BASE
           MSI        A4,R3                .     ADDRESS
           AH         A3,A4                .     FOR Y
                                  BEGIN LOOP TO FORM INNER PRODUCT
LOOP       FEL        A4,0,*A2 . LOAD X, CONVERT TO DOUBLE, AND INC. INDEX
           FEL        A6,0,*A3 . LOAD Y, CONVERT TO DOUBLE, AND INC. INDEX
           DFM        A4,A6                . MULTIPLY X TIMES Y
           DFA        A0,A4                . ACCUMULATE INNER PRODUCT
           JGD        R3,LOOP              . END OF INNER PRODUCT LOOP
           DL         A6,SAVE              . RESTORE REGISTERS A6 AND A7
           J          6,X11                . RETURN FOR N.GT.0

$(0)
SAVE       +          0D                   . PLACE TO SAVE A6 AND A7
           END .
```

Figure B-2. UNIVAC 1108 Assembly Code for DSDOT

```
******************************************************************
*       DOUBLE PRECISION FUNCTION DSDOT (N,X,INCX,Y,INCY)       *
*                                                                *
*       COMPUTED AS SUM FROM I =1 TO N OF A(I)*B(I)              *
*       WHERE X( ) AND Y( ) ARE OF TYPE REAL WITH               *
*                                                                *
*       X(1-INCX+I*INCX)   = A(I) IF INCX .GT. 0                 *
*                                                                *
*       X(1-N*INCX+I*INCX) = A(I) IF INCX .LE. 0                 *
*                                                                *
*       SIMILAR DEFINITIONS FOR Y( ) AND B(I)                    *
*                                                                *
*       THE SUM IS ACCUMULATED IN DOUBLE PRECISION               *
*                                                                *
*       WRITTEN BY R. J. HANSON, 27 JULY 1973                    *
******************************************************************
*
DSDOT    CSECT
         USING *,15              BASE REGISTER ASSIGNMENT
         SAVE  (1,9),,*          SAVE USED REGISTER CONTENTS
         LM    2,6,0(1)          GET ALL POINTERS TO ARGS. IN REGS. 2-6
         SDR   0,0               SET DSDOT = 0
         L     7,0(2)            GET N AND
         LTR   7,7               (SET CONDITION CODES) AND SEE
         BNP   DONE              IF IT'S .LE. 0.  DEFINE DSDOT = 0 IF SO.
         L     2,7               SAVE N FOR COUNTING
         BCTR  7,0               COMPUTE N-1
         L     6,0(6)            GET INCY AND SEE
         SLA   6,2               (SET CONDITION CODES AND MULTIPLY * 4)
         BP    INCXT             IF IT'S .GT. 0
         LR    9,6               GET INCY * 4 AND
         MR    8,7               COMPUTE INCY*(N-1)*4 AND
         SR    5,9               FIX BASE ADDRESS OF Y( )
INCXT    L     4,0(4)            GET INCX AND SEE
         SLA   4,2               (SET CONDITION CODES AND MULTIPLY * 4)
         BP    LOOP              IF IT'S .GT. 0
         LR    9,4               GET INCX * 4 AND
         MR    8,7               COMPUTE INCX*(N-1)*4 AND
         SR    3,9               FIX BASE ADDRESS OF X( )
LOOP     LE    2,0(0,3)          GET A(I) AND
         ME    2,0(0,5)          MULTIPLY BY B(I) AND
         ADR   0,2               ACCUMULATE INNER PRODUCT
         AR    3,4               THEN FIX THE
         AR    5,6               ADDRESSES AND CHECK FOR
         BCT   2,LOOP            END OF THE LOOP
DONE     RETURN (1,9),T          GO BACK TO THE CALLING PROGRAM
         END
```

Figure B-3. IBM 360/75 Assembly Code for DSDOT

REFERENCES

1. Krogh, F. T., On the Use of Assembly Code for Heavily Used Modules in Linear Algebra. Section 914 Internal Technical Memorandum No. 203, May 2, 1972. (JPL internal document.)

2. Gentleman, M. W., (1972A) Least Squares Computations by Givens Transformations Without Square Roots, University of Waterloo Report CSSR-2062, 17 pp.

3. Gentleman, W. M., (1972B) Basic Procedures for Large, Sparse or Weighted Linear Least Squares Problems, University of Waterloo Report CSSR-2068, 14 pp.

NASA — JPL — Coml., L.A., Calif.