AD-A054 358 TECHNOLOGY SERVICE CORP SANTA MONICA CALIF F/G 17/9
MULTIDOMAIN ALGORITHM EVALUATION. VOLUME II.(U)
APR 78 W C LILES, J C DEMMEL, I S REED F30602-76-C-0319
UNCLASSIFIED TSC-PD-8525-1-VOL-2 RADC-TR-78-59-VOL-2 NL

1 OF 3
AD A054358

RADC-TR-78-59, Volume II (of two)
Final Technical Report
April 1978

MULTIDOMAIN ALGORITHM EVALUATION

William C. Liles
James C. Demmel
Irving S. Reed
John D. Mallett
Lawrence E. Brennan

Technology Service Corporation

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-59, Volume II (of two) has been reviewed and is approved for publication.

APPROVED: Vincent Vannicola

VINCENT VANNICOLA
Project Engineer

APPROVED: Joseph L Ryerson

JOSEPH L. RYERSON
Technical Director
Surveillance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (OCTS) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-78-59, Vol II (of two) - Vol-2
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): MULTIDOMAIN ALGORITHM EVALUATION, Volume II.
5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, 25 Jun 76 - 18 Oct 77
6. PERFORMING ORG. REPORT NUMBER: TSC-PD-B525-1 - Vol-2
7. AUTHOR(s): William C. Liles, James C. Demmel, Irving S. Reed, John D. Mallett, Lawrence E. Brennan
8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0319
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Technology Service Corporation, 2811 Wilshire Boulevard, Santa Monica CA 90403
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 45060196
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (OCTS), Griffiss AFB NY 13441
12. REPORT DATE: April 1978
13. NUMBER OF PAGES: 273
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
Vincent Vannicola (OCTS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Adaptive array radar
Adaptive algorithms
Associative processors
Parallel processors
Vector pipeline processors
Matrix inversion

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of this study is to evaluate different algorithms for solving for up to 200 adaptive weights in an adaptive array radar, using the sample covariance matrix inversion technique. The sample covariance matrix inversion technique was studied because of its ability to handle adaptation in many domains, i.e., spatial, temporal, and polarization.

(Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

404 432 LB

Item 20 (Cont'd)

The algorithms and their implementations on different computer architectures, such as associative, parallel, vector pipeline, and sequential, are considered. Both theoretical timings and actual timings on currently available machines are obtained.

Major conclusions reached are that 1) because of the strong dependence of an algorithm's implementation on the computer architecture, it is not possible to choose the best algorithm by operation counts; 2) it is possible to greatly improve system performance by using separate processors to perform the covariance matrix computation and weight calculations; 3) parallel complex arithmetic implemented in hardware would greatly improve a system's performance; and 4) associativity is not useful for this problem.

Finally, an architecture designed specifically for solving for adaptive weights is outlined.

TABLE OF CONTENTS

Appendix A. Implementations of Algorithms for Computing the Sample Covariance Matrix on a Vector Pipeline Processor

All the implementations are written in an easily understood, structured programming language. These implementations are presented for operation count purposes only and are not intended to be compilable code. For a discussion of these implementations, please see Section 4.3.2.2 "Calculating the Sample Covariance Matrix on a Vector Pipeline Processor."

In the following discussion, sample covariance matrix will be abbreviated to SCM and sample vector to SV. MR and MI will be the arrays containing, respectively, the real and imaginary parts of the SCM and will be floating-point (FP). XR and XI are FP arrays containing the real and imaginary parts of the SV. In front of each implementation will be a list of variables with their types (floating-point (FP) or integer (I)), their lengths (as a function of the number of weights (N) and the number of samples (NS)), and their contents and storage scheme (contents omitted for MR, MI, XR, XI; storage scheme is vectorwise, componentwise, etc.).

Vectors are denoted by double subscripts like MR(K,L), which means the L-word-long vector starting at location K of array MR. Vector operations are denoted by

1) A(K1,L1) = B(K2,L1) ± C(K3,L1)

2) A(K1,L2) = D(K2,L2)

3) X = SUM[A(K2,L3)]

which are equivalent to, respectively,

1) FOR LOC=0 to L1-1

        A(K1 + LOC) = B(K2 + LOC) ± C(K3 + LOC)

   END FOR

```
2)  FOR LOC = 0 to L2-1
        A(K1 + LOC) = D(K2 + LOC)
    END FOR
3)  X = 0
    FOR LOC = 0 to L3-1
    X = X + A(K2 + LOC)
    END FOR
```

If a non-vector appears to the right of the equal sign in case 1) or 2), it is treated as though it were a vector of the appropriate length containing the constant value of the non-vector.

FOR-END FOR loops behave as they should: if the initial value is greater than the final value and the step-size is positive (default = 1), or if the initial value is less than the final value and the step-size is negative, the loop is skipped and control passes to the first statement following the END FOR.

Implementation 1--Unpacked Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N^2	rowwise
MI	FP	N^2	rowwise
XR	FP	N	
XI	FP	N	
ROW	I	1	current row of SCM
START	I	1	starting location of current row of SCM

```
      BEGIN
      START = 1
C     UPDATE EACH ROW OF THE MATRIX
      FOR ROW = 1 TO N
C         UPDATE THE REAL PART
          MR(START,N) = MR(START,N)  +  XR(ROW)*XR(1,N) + XI(ROW)*XI(1,N)
C         UPDATE THE IMAGINARY PART
          MI(START,N) = MI(START,N)  +  XR(ROW)*XI(1,N) - XI(ROW)*XR(1,N)
          START = START + N
       END FOR
       END
```

Implementation 1--Unpacked Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
XR	FP	N	
XI	FP	N	
ROW	I	1	current row of SCM
START	I	1	starting location of current row of SCM
LENGTH	I	1	length of current row of SCM

```
      BEGIN
      START = 1
      LENGTH = N
C     UPDATE THE 1st THROUGH N-2nd ROWS OF THE MATRIX
      FOR ROW = 1 TO N-2
C         UPDATE THE REAL PART
          MR(START,LENGTH) = MR(START,LENGTH) + XR(ROW)*
               XR(ROW,LENGTH) + XI(ROW)*XI(ROW,LENGTH)
C         UPDATE THE IMAGINARY PART
          MI(START + 1,LENGTH - 1) = MI(START + 1,LENGTH - 1)
             + XR(ROW)*XI(ROW + 1,LENGTH - 1)
             - XI(ROW)*XR(ROW + 1,LENGTH -1)
          START = START + LENGTH
          LENGTH = LENGTH - 1
      END FOR
C         UPDATE THE REAL PART OF THE N-1st ROW
      MR(START,2) = MR(START,2)
            + XR(N-1)*XR(N-1,2) + XI(N-1)*XI(N-1,2)
C     UPDATE THE IMAGINARY PART OF THE N-1st ROW
      MI(START+1) = MI(START+1) + XR(N-1)*XI(N) - XI(N-1)*XR(N)
C     UPDATE THE REAL PART OF THE Nth ROW
      MR(START+2) = MR(START+2) + XR(N)*XR(N) + XI(N)*XI(N)
      END
```

Implementation 2--Unpacked Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N^2	rowwise
MI	FP	N^2	rowwise
XR	FP	N	
XI	FP	N	
TR1	FP	N^2	temporary array containing a stretched form of XR
TR2	FP	N^2	temporary array containing a stretched form of XR
TI1	FP	N^2	temporary array containing a stretched form of XI
TI2	FP	N^2	temporary array containing a stretched form of XI
ROW	I	1	current row of SCM being stretched
START	I	1	starting location of current row of SCM
LENGTH	I	1	length of whole SCM

```
      BEGIN
C     STRETCH OUT SAMPLE VECTORS
      START = 1
      FOR ROW = 1 to N
C         STRETCH THE REAL PARTS
          TR1(START,N) = XR(1,N)
          TR2(START,N) = XR(ROW)
C         STRETCH THE IMAGINARY PARTS
          TI1(START,N) = XI(1,N)
          TI2(START,N) = XI(ROW)
          START = START + N
      END FOR
C     UPDATE MATRIX
      LENGTH = N*N
      MR(1,LENGTH) = MR(1,LENGTH) + TR1(1,LENGTH)*TR2(1,LENGTH)
                        +TI1(1,LENGTH)*TI2(1,LENGTH)
      MI(1,LENGTH) = MI(1,LENGTH) + TR1(1,LENGTH)* TI2(1,LENGTH)
                        - TI1(1,LENGTH)*TR2(1,LENGTH)
      END
```

Implementation 2--Packed Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
XR	FP	N	
XI	FP	N	
TR1	FP	N(N+1)/2	temporary array containing a stretched form of XR
TR2	FP	N(N+1)/2	temporary array containing a stretched form of XR
TI1	FP	N(N+1)/2	temporary array containing a stretched form of XI
TI2	FP	N(N+1)/2	temporary array containing a stretched form of XI
ROW	I	1	current row of SCM being stretched
START	I	1	starting location of current row of SCM
LENGTH	I	1	length of current row of SCM, then length of whole SCM

```
      BEGIN
C     STRETCH OUT SAMPLE VECTORS
      START = 1
      LENGTH = N
      FOR ROW = 1 to N-1
C           STRETCH OUT REAL PARTS
            TR1(START,LENGTH) = XR(ROW,LENGTH)
            TR2(START,LENGTH) = XR(ROW)
C           STRETCH OUT IMAGINARY PARTS
            TI1(START,LENGTH) = XI(ROW,LENGTH)
            TI2(START,LENGTH) = XI(ROW)
            START = START + LENGTH
            LENGTH = LENGTH - 1
      END FOR
      TR1(START) = XR(N)
      TR2(START) = XR(N)
      TI1(START) = XI(N)
      TI2(START) = XI(N)
C     UPDATE MATRIX
      LENGTH = N*(N+1)/2
      MR(1,LENGTH) = MR(1,LENGTH) + TR1(1,LENGTH)*
           TR2(1,LENGTH) + TI1(1,LENGTH)*TI2(1,LENGTH)
      MI(1,LENGTH) = MI(1,LENGTH) + TR1(1,LENGTH)* TI2(1,LENGTH)
                       - TI1(1,LENGTH)*TR2(1,LENGTH)
      END
```

Implementation 3--Unpacked Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N^2	rowwise
MI	FP	N^2	rowwise
XR	FP	N*NS	real parts of NS SV's stored componentwise (i.e., 1st component of all NS samples, followed by 2nd component of all NS samples, etc.)
XI	FP	N*NS	imaginary part of NS SV's stored componentwise
T	FP	2*NS	temporary array to be used with SUM
ROW	I	1	row of current element of SCM
COLUMN	I	1	column of current element of SCM
STARTM	I	1	location of current element of SCM
STARTMT	I	1	location of transpose of current element of SCM
STARTXR	I	1	starting location of ROWth components of SV's
STARTXC	I	1	starting locations of COLUMNth components of SV's
NS2	I	1	2* number of SV's

```
      BEGIN
C     COMPUTE THE INNER PRODUCT OF THE DATA FOR
C     EACH ELEMENT OF THE MATRIX
      NS2 = 2*NS
      STARTM  = 1
      STARTXR = 1
      STARTXC = 1
      FOR ROW = 1 to N-1
C           COMPUTE DIAGONAL ELEMENT
            T(1,NS) = XR(STARTXR,NS)*XR(STARTXC,NS)
            T(NS+1,NS) = XI(STARTXR,NS)*XI(STARTXC,NS)
            MR(STARTM) = SUM(T(1,NS2))
            STARTMT = STARTM + N
            STARTM = STARTM + 1
C           COMPUTE OTHER ELEMENTS
            FOR COLUMN = ROW + 1 to N
C                 STARTXC = STARTXC + NS
                  CALCULATE REAL PART
                  T(1,NS) = XR(STARTXR,NS)*XR(STARTXC,NS)
                  T(NS+1,NS) = XI(STARTXR,NS)*XI(STARTXC,NS)
                  MR(STARTM) = SUM(T(1,NS2))
                  MR(STARTMT) = MR(STARTM)
C                 CALCULATE IMAGINARY PART
                  T(1,NS) = XR(STARTXR,NS)*XI(STARTXC,NS)
                  T(NS+1,NS) = -XI(STARTXR,NS)*XR(STARTXC,NS)
                  MI(STARTM) = SUM(T(1,NS2))
                  MI(STARTMT) = -MI(STARTM)
                  STARTMT = STARTMT + N
                  STARTM = STARTM + 1
            END FOR
            STARTM = STARTM + ROW
            STARTXR = STARTXR + NS
            STARTXC = STARTXR
      END FOR
C     COMPUTE LAST DIAGONAL ELEMENT
      T(1,NS) = XR(STARTXR,NS)*XR(STARTXR,NS)
      T(NS+1,NS) = XI(STARTXR,NS)*XI(STARTXR,NS)
      MR(STARTM) = SUM(T(1,NS2))
      END
```

Implementation 3--Packed Storage

VARIABLE	TYPE	LENGTH	CONTENTS
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
XR	FP	N*NS	real parts of NS SV's stored componentwise (i.e., the 1st component of all NS SV's, followed by the 2nd component of all SV's, etc.)
XI	FP	N*NS	imaginary part of NS SV's stored componentwise
T	FP	NS	temporary array used with SUM
ROW	I	1	row of current element of SCM
COLUMN	I	1	column of current element of SCM
STARTM	I	1	location of current element of SCM
STARTXR	I	1	starting location of ROWth components of SV's
STARTXC	I	1	starting location of *COLUMNth* components of SV's
NS2	I	1	2* number of SV's

```
      BEGIN
C     COMPUTE THE INNER PRODUCT OF THE DATA FOR EACH
C     ELEMENT OF THE MATRIX
      NS2 = 2*NS
      STARTM = 1
      STARTXC = 1
      STARTXR = 1
      FOR ROW = 1 to N-1
C          COMPUTE DIAGONAL ELEMENT
           T(1,NS) = XR(STARTXR,NS)*XR(STARTXC,NS)
           T(NS+1,NS) = XI(STARTXR,NS)*XI(STARTXC,NS)
           MR(STARTM) = SUM(T(1,NS2))
           STARTM = STARTM + 1
C          COMPUTE OTHER ELEMENTS
           FOR COLUMN = ROW + 1 to N
                STARTXC = STARTXC + NS
C               CALCULATE REAL PART
                T(1,NS) = XR(STARTXR,NS)*XR(STARTXC,NS)
                T(NS+1,NS) = XI(STARTXR,NS)*XI(STARTXC,NS)
                MR(STARTM) = SUM(T(1,NS2))
C               CALCULATE IMAGINARY PART
                T(1,NS) = XR(STARTXR,NS)*XI(STARTXC,NS)
                T(NS+1,NS) = -XI(STARTXR,NS)*XR(STARTXC,NS)
                MI(STARTM) = SUM(T(1,NS2))
                STARTM = STARTM + 1
           END FOR
           STARTXR = STARTXR + NS
           STARTXC = STARTXR
      END FOR
C     COMPUTE LAST DIAGONAL ELEMENT
      T(1,NS) = XR(STARTXR,NS)*XR(STARTXR,NS)
      T(NS+1,NS) = XI(STARTXI,NS)*XI(STARTXI,NS)
      MR(STARTM) = SUM(T(1,NS2))
      END
```

Appendix B. Implementations of Algorithms for Direct Methods of Solving $MW = \overline{S}$ on Vector Pipeline Processors

For an explanation of the symbols and abbreviations used in this appendix, please see the introduction to Appendix A and the discussion of Section 4.3.2.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

Gauss-Jordon (GJ)

Variables	Type	Length	Contents
MR	FP	N*(N+K)	the i^{th} group of N+K words contains the real part of the i^{th} row of the SCM followed by the i^{th} real components of K steering vectors.
MI	FP	N*(N+K)	imaginary counterpart of MR
SR			} contained in MR and MI
SI			
ROW	I	1	current row of SCM
DIVID	RP	1	scale factor to normalize ROWth row of SCM
SUBROW	I	1	row from which multiple of current row is subtracted
STARTR	I	1	ROW+1st location of ROWth row of SCM
STARTS	I	1	ROW+1st location of SUBROWth row of SCM
LENGTH	I	1	length of current row of SCM
DELTA	I	1	N+K; used to update STARTR and STARTS

```
      BEGIN
      DELTA = N+K
      LENGTH = DELTA
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO ONLY ROWTH ROW HAS 1 IN IT,
C     AND THE REST HAVE ZEROS

      FOR ROW = 1 TO N
      .   LENGTH = LENGTH-1
      .   DIVID = 1./MR(STARTR-1)

C     *   PUT A 1 IN THE ROW-TH ROW

      .   MR(STARTR,LENGTH) = MR(STARTR,LENGTH)*DIVID
      .   MI(STARTR,LENGTH) = MI(STARTR,LENGTH)*DIVID
      .   STARTS = ROW+1

C     *   ELIMINATE FROM THE OTHER ROWS

      .   FOR SUBROW = 1 TO N
      .   .   IF SUBROW .NE. ROW
      .   .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .   .                     - MR(STARTS-1)*MR(STARTR,LENGTH)
     +.   .   .                     + MI(STARTS-1)*MI(STARTR,LENGTH)
      .   .   .   MI(STARTS,LENGTH) = MI(STARTS,LENGTH)
     +.   .   .                     - MR(STARTS-1)*MI(STARTR,LENGTH)
     +.   .   .                     - MI(STARTS-1)*MR(STARTR,LENGTH)
      .   .   END IF
      .   .   STARTS = STARTS+DELTA
      .   END FOR
      .   STARTR = STARTR+DELTA+1
      END FOR
      END
     (NOTE THAT IF K=1 THE VECTOR OPERATIONS WHEN ROW = N WILL
      BE ONE SET OF OPERANDS LONG AND SHOULD BE CHANGED TO
      SCALAR OPERATIONS, THIS IMPLEMENTATION ASSUMES K > 1)
```

Gaussian Elimination-GE-Decomposition

Variable	Type	Length	Contents
MR	FP	N^2	rowwise - unpacked
MI	FP	N^2	rowwise - unpacked
RECIP	FP	N	reciprocals of components of D in decomposition $M = LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st word of ROWth row
STARTS	I	1	location of ROW+1st word of SUBROWth row
LENGTH	I	1	length of ROWth row starting in column ROW+1

```
      BEGIN
      LENGTH = N
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO IT HAS ZEROES BELOW A UNIT DIAGONAL

      FOR ROW = 1 TO N-2
      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   LENGTH = LENGTH-1

C     *   MAKE UNIT DIAGONAL

      .   MR(STARTR,LENGTH) = RECIP(ROW) * MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW) * MI(STARTR,LENGTH)

C     *   ELIMINATE BELOW THE DIAGONAL

      .   STARTS = STARTR
      .   FOR SUBROW = ROW  + 1 TO N
      .   .   STARTS = STARTS + N
      .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .                   -MR(STARTS-1) * MR(STARTR,LENGTH)
     +.   .                   +MI(STARTS-1)*MI(STARTR,LENGTH)
      .   .   MI(STARTS,LENGTH) = MI(STARTS,LENGTH)
     +.   .                   -MR(STARTS-1) * MI(STARTR,LENGTH)
     +.   .                   -MI(STARTS-1) * MR(STARTR,LENGTH)
      .   END FOR
      .   STARTR = STARTR + N + 1
      END FOR

C     ELIMINATE N-1ST COLUMN

      RECIP(N-1) = 1./MR(STARTR-1)
      MR(STARTR) = RECIP(N-1) * MR(STARTR)
      MI(STARTR) = RECIP(N-1) * MI(STARTR)

C     CALCULATE RECIPROCAL OF LAST DIAGONAL ELEMENT

      STARTS = STARTR + N
      RECIP(N) = 1./(MR(STARTS) - MR(STARTR) * MR(STARTS-1)
     *             +MI(STARTR) * MI(STARTS -1))
      END
```

Cholesky without square roots-LDL*-Packed decomposition (the only difference between the packed and unpacked version is the calculation of subscripts and pointers, so numbers and expressions in brackets refer to the unpacked version and replace the numbers and expressions they follow).

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	rowwise
MI	FP	N(N+1)/2 [N^2]	rowwise
TR	FP	N	temporary array
TI	FP	N	temporary array
RECIP	FP	N	reciprocals of elements of D in decomposition M = LDL*
TR1	FP	1	temporary location
TI1	FP	1	temporary location
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at element ROW+1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW

```
      BEGIN
      LAST = N*(N+1)/2 [LAST = N * N]
      STARTR = 2
      LENGTH = N

C      CALCULATE ROWTH ROW OF L* FACTOR IN M = LDL*

      FOR ROW = 1 TO N-2

C      *   MAKE UNIT DIAGONAL, SAVE RECIPROCAL

      .   RECIP(ROW) = 1./MR(STARTR - 1)
      .   LENGTH = LENGTH - 1
      .   TR(ROW + 1,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   TI(ROW+1,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C      *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N]
      .   FOR SUBROW = ROW + 1 TO N - 2
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .                 -TR(SUBROW)*MR(STARTRS,LENGTHS)
     +.   .                 -TI(SUBROW)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS + 1,LENGTHS-1) = MI(STARTS + 1,LENGTHS - 1)
     +.   .                 -TR(SUBROW)*MI(STARTRS + 1,LENGTHS - 1)
     +.   .                 +TI(SUBROW)*MR(STARTRS + 1, LENGTHS - 1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + 1]
      .   .   LENGTHS = LENGTHS - 1
      .   END FOR

C      *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   MR(STARTS,2) = MR(STARTS,2)-TR(N-1)*MR(STARTRS,2)
     +.       - TI(N-1)*MI(STARTRS,2)
      .   MI(STARTS + 1) = MI(STARTS + 1) - TR(N-1)*MI(STARTRS + 1)
     +.       + TI(N-1)*MR(STARTRS + 1)
      .   STARTRS = STARTRS + 1

C      *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   MR(LAST) = MR(LAST) - TR(N)*MR(STARTRS) -
     +.          TI(N)*MI(STARTRS)

C      *   MOVE NORMALIZED ROW FROM MR AND MI TO MR AND MI

      .   MR(STARTR,LENGTH) = TR(ROW + 1, LENGTH)
      .   MI(STARTR,LENGTH) = TI(ROW + 1,LENGTH)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]
      END FOR

C      CALCULATE N-1ST ROW

      RECIP(N-1) = 1./MR(STARTR - 1)
      TR1 = RECIP(N-1)*MR(STARTR)
      TI2 = RECIP(N-1)*MI(STARTR)

C      CALCULATE LAST DIAGONAL RECIPROCAL

      RECIP(N) = 1./(MR(LAST) - TR1*MR(STARTR) -
     +       TI1*MI(STARTR))
      MR(STARTR) = TR1
      MI(STARTR) = TI2
      END
```

Cholesky with square roots-LL^*- Packed decomposition (see comments for LDL^* decomposition)

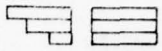
Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	rowwise
MI	FP	$N(N+1)/2$ [N^2]	rowwise
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition $M=LL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at column ROW+1
LENGTHS	I	1	length of SUBROWth row starting in column SUBROW

```
      BEGIN
      LAST = N*(N+1)/2 [LAST = N*N]
      STARTR = 2
      LENGTH = N

C     CALCULATE ROWTH ROW OF L* FACTOR IN H = LL*

      FOR ROW = 1 TO N - 2

C     *   CALCULATE RECIPROCAL DIAGONAL AND ROW

      .   RECIP(ROW) = 1./SQRT (HR(STARTR-1))
      .   LENGTH = LENGTH -1
      .   HR(STARTR,LENGTH) = RECIP(ROW)*HR(STARTR,LENGTH)
      .   HI(STARTR,LENGTH) = RECIP(ROW)*HI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N]
      .   FOR  SUBROW = ROW + 1 TO N-2
      .   .   HR(STARTS,LENGTHS) = HR(STARTS,LENGTHS)
     +.   .               -HR(STARTRS)*HR(STARTRS,LENGTHS)
     +.   .               -HI(STARTRS)*HI(STARTRS,LENGTHS)
      .   .   HI(STARTS + 1,LENGTHS - 1) = HI(STARTS + 1, LENGTHS - 1)
     *.   .               -HR(STARTRS)*HI(STARTRS + 1, LENGTHS - 1)
     *.   .               +HI(STARTRS)*HR(STARTRS + 1, LENGTHS -1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + 1]
      .   .   LENGTHS = LENGTHS - 1
      .   END FOR

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   HR(STARTS,2) = HR(STARTS,2) - HR(STARTRS)*
     +.        HR(STARTRS,2) - HI(STARTRS)*HI(STARTRS,2)
      .   HI(STARTS + 1) = HI(STARTS + 1) - HR(STARTRS)*
     +.        HI(STARTRS + 1) + HR(STARTRS)*HR(STARTRS + 1)
      .   STARTRS = STARTRS + 1

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   HR(LAST) = HR(LAST) - HR(STARTRS)*HR(STARTRS)
     *.              -HI(STARTRS)*HI(STARTRS)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]
      END FOR

C     CALCULATE N-1ST ROW

      RECIP (N-1) = 1./SQRT (HR(STARTR - 1))
      HR(STARTR) = RECIP (N-1)*HR(STARTR)
      HI(STARTR) = RECIP (N-1)*HI(STARTR)

C     CALCULATE NTH DIAGONAL RECIPROCAL

      RECIP(N) = 1./SQRT (HR(LAST)-HR(STARTR)*HR(STARTR)
     +            -HI(STARTR)*HI(STARTR))
      END
```

Gaussian Elimination with Vectorwise Augmentation-GE- 

Variable	Type	Length	Contents
MR	FP	N^2	rowwise
MI	FP	N^2	rowwise
SR	FP	KN	real part of K steering vectors vectorwise
SI	FP	KN	imaginary part of K steering vectors vectorwise
RECIP	FP	N	reciprocals of components of D in decomposition $M=LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st word of ROWth row
STARTS	I	1	location of ROW+1st word of SUBROWth row starting at column ROW+1
LENGTH	I	1	length of ROWth row
AUG	I	1	current steering vector
STARTSV	I	1	location of ROW+1st element of current steering vector
STARTSVM	I	1	location of first element of current steering vector

```
      BEGIN
      LENGTH = N
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO IT HAS ZEROES BELOW A UNIT DIAGONAL

      FOR ROW = 1 TO N - 2
      .       RECIP(ROW) = 1./MR(STARTR - 1)
      .       LENGTH = LENGTH - 1

C     *   MAKE UNIT DIAGONAL

      .   MR(STARTR,LENGTH) = RECIP(ROW) * MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW) * MI(STARTR,LENGTH)

C     *   ELIMINATE BELOW THE DIAGONAL

      .   STARTS = STARTR
      .   FOR SUBROW = ROW + 1 TO N
      .   .   STARTS = STARTS + N
      .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .         -MR(STARTS - 1) * MR(STARTR,LENGTH)
     +.   .          + MI(STARTS - 1)* MI(STARTR,LENGTH)
      .   .   MI(STARTS,LENGTH) = MI(STARTS,LENGTH)
     +.   .          -MR(STARTS-1) * MI(STARTR,LENGTH)
     +.   .          - MI(STARTS - 1) * MR(STARTR,LENGTH)
      .   END FOR

C     *   ELIMINATE FROM STEERING VECTORS

      .   STARTSV = ROW + 1
      .   FOR AUG = 1 TO K
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .          - SR(STARTSV-1) * MR(STARTR,LENGTH)
     +.   .          -SI(STARTSV-1) * MI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .         +SR(STARTSV-1) * MI(STARTR,LENGTH)
     +.   .          -SI(STARTSV-1) * MR(STARTR,LENGTH)
      .   .   STARTSV = STARTSV + N
      .   END FOR
      .   STARTR = STARTR + N + 1
      END FOR

C     ELIMINATE N-1ST COLUMN

      RECIP(N-1) = 1./MR(STARTR-1)
      MR(STARTR) = RECIP(N-1) * MR(STARTR)
      MI(STARTR) = RECIP (N-1) * MI(STARTR)

C     CALCULATE RECIPROCAL OF LAST DIAGONAL ELEMENT

      STARTS = STARTR + N
      RECIP(N) = 1./(MR(STARTS) - MR(STARTR) * MR(STARTS-1)
     +       *MI(STARTR) * MI(STARTS-1))

C     ELIMINATE FROM STEERING VECTORS AND DIVIDE BY DIAGONAL ELEMENTS

      STARTSV = N
      STARTSV1 = 1
      FOR AUG = 1 TO K
      .   SR(STARTSV) = SR(STARTSV) - SR(STARTSV-1) * MR(STARTR)
     +.          -SI(STARTSV-1) * MI(STARTR)
      .   SI(STARTSV) = SI(STARTSV) + SR(STARTSV-1) * MI(STARTR)
     +.          -SI(STARTSV-1) * MR(STARTR)
      .   SR(STARTSV1,N) = RECIP(1,N) * SR(STARTSV1,N)
      .   SI(STARTSV1,N) = RECIP(1,N) * SI(STARTSV1,N)
      .   STARTSV = STARTSV + N
      .   STARTSV1 = STARTSV1 + 1
      END FOR
      END
```

Gaussian Elimination with Componentwise Augmentation - GE - 

Variable	Type	Length	Contents
MR	FP	N·(N + K)	the i^{th} group of N+K words contains the real part of the i^{th} row of the SCM followed by the real parts of the i^{th} components of the K steering vectors.
MI	FP	N·(N + K)	analogous to MR, except contains imaginary parts
SR			contained in MR and MI
SI			
RECIP	FP	N	reciprocals of elements of D in decomposition $M = LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st word of ROWth row
STARTS	I	1	location of ROW+1st word of SUBROWth row
LENGTH	I	1	length of ROWth row starting in column ROW+1

```
      BEGIN
      LENGTH = N + K
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO IT HAS ZEROES BELOW A UNIT DIAGONAL

      FOR ROW = 1 TO N - 1
      .   RECIP(ROW) = 1./MR(STARTR - 1)
      .   LENGTH = LENGTH - 1

C     *   MAKE UNIT DIAGONAL

      .   MR(STARTR,LENGTH) = RECIP(ROW) * MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW) * MI(STARTR,LENGTH)

C     *   ELIMINATE BELOW THE DIAGONAL

      .   STARTS = STARTR
      .   FOR SUBROW = ROW + 1 TO N
      .   .   STARTS = STARTS + N + K
      .   .      MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .                -MR(STARTS-1) * MR(STARTR,LENGTH)
     +.   .                +MI(STARTS-1) * MI(STARTR,LENGTH)
      .   .   MI(STARTS,LENGTH) = MI(STARTS,LENGTH)
     +.   .                -MR(STARTS-1) * MI(STARTR,LENGTH)
     +.   .                -MI(STARTS-1) * MR(STARTR,LENGTH)
      .   END FOR
      .   STARTR = STARTR + N + K + 1
      END FOR
      RECIP(N) = 1./MR(STARTR-1)
      MR(STARTR,K) = RECIP(N) * MR(STARTR,K)
      MI(STARTR,K) =  RECIP(N) * MI(STARTR,K)

C

      END
C

(NOTE THAT IF K = 1, THEN THE FINAL 2 VECTOR
OPERATIONS WILL ONLY BE ONE SET OF OPERANDS
LONG, AND SHOULD BE CHANGED TO SCALAR OPERATIONS.)
```

Cholesky without Square Roots with Vectorwise Augmentation-LDL* - 

(see comments for LDL* Decomposition)

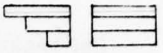
Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	rowwise
MI	FP	$N(N+1)/2$ [N^2]	rowwise
SR	FP	K·N	real parts of K steering vectors vectorwise
SI	FP	K·N	imaginary parts of K steering vectors vectorwise
TR	FP	N	temporary array
TI	FP	N	temporary array
RECIP	FP	N	reciprocals of elements of D in decomposition M = LDL*
TR1	FP	1	temporary location
TI1	FP	1	temporary location
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row

Cholesky without Square Roots with Vectorwise Augmentation-LDL* - 
(see comments for LDL* Decomposition) (Cont'd)

Variable	Type	Length	Contents
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at element ROW+1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW
AUG	I	1	current steering vector
STARTSV	I	1	location of ROW+1st element of current steering vector
STARTSVM	I	1	location of first element of current steering vector

```
      BEGIN
      LAST = N*(N+1)/2  [LAST = N*N]
      STARTR = 2
      LENGTH = N

C     CALCULATE ROWTH ROW OF L* FACTOR IN M = LDL*

      FOR ROW = 1 TO N-2

C     *   MAKE UNIT DIAGONAL, SAVE RECIPROCAL

      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   LENGTH = LENGTH-1
      .   TR(ROW+1,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   TI(ROW+1,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N]
      .   FOR SUBROW = ROW + 1 TO N-2
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .         -TR(SUBROW)*MR(STARTRS,LENGTHS)
     +.   .         -TI(SUBROW)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS+1 ,LENGTHS-1) = MI(STARTS + 1,LENGTHS - 1)
     +.   .         -TR(SUBROW)*MI(STARTRS + 1,LENGTHS - 1)
     +.   .         +TI(SUBROW)*MR(STARTRS + 1 ,LENGTHS - 1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + 1]
      .   .   LENGTHS = LENGTHS - 1
      .   END FOR

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   MR(STARTS,2) = MR(STARTS,2)-TR(N-1)*MR(STARTRS,2)
     +.             -TI(N-1)*MI(STARTRS,2)
      .   MI(STARTS + 1) = MI(STARTS + 1) -TR(N-1)*MI(STARTRS + 1)
     +.        +TI(N-1)*MR(STARTRS + 1)
      .   STARTRS = STARTRS + 1

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   MR(LAST) = MR(LAST) - TR(N)*MR(STARTRS)
     +.        -TI(N)*MI(STARTRS)

C     *   MOVE NORMALIZED ROW FROM TR AND TI TO MR AND MI

      .   MR(STARTR,LENGTH) = TR(ROW + 1,LENGTH)
      .   MI(STARTR,LENGTH) = TI(ROW + 1,LENGTH)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]

C     *   SUBTRACT MULTIPLES OF CURRENT ROW FROM STEERING VECTORS

      .   STARTSV = ROW + 1
      .   FOR AUG = 1 TO K
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .         -SR(STARTSV-1)*TR(ROW+1,LENGTH)
     +.   .         -SI(STARTSV-1)*TI(ROW+1,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .               +SR(STARTSV-1)*TI(ROW+1,LENGTH)
     +.   .               -SI(STARTSV-1)*TR(ROW+1,LENGTH)
      .   .   STARTSV = STARTSV + N
      .   END FOR
      END FOR
```

```
C       CALCULATE N-1ST ROW

        RECIP(N-1) = 1./MR(STARTR-1)
        TR1 = RECIP (N-1)*MR(STARTR)
        TR2 = RECIP(N-1)*MI(STARTR)

C       CALCULATE NTH DIAGONAL RECIPROCAL

        RECIP(N) = 1./(MR(LAST) - TR1 * MR(STARTR)
       +           -TI1*MI(STARTR))
        MR(STARTR) = TR1
        MI(STARTR) = TI1
        STARTSV = N
        STARTSVM = 1
        FOR AUG = 1 TO K
        .    SR(STARTSV) = SR(STARTSV) - SR(STARTSV-1) * TR1
       +.    -SI(STARTSV-1) * TI1
        .    SI(STARTSV) = SI(STARTSV) + SR(STARTSV - 1) * TI1
       +.    -SI(STARTSV-1) * TR1
        .    SR(STARTSVM,N) = RECIP(1,N)*SR(STARTSVM,N)
        .    SI(STARTSVM,N) = RECIP(1,N)*SI(STARTSVM,N)
        .    STARTSV = STARTSV + N
        .    STARTSVM = STARTSVM + N
        END FOR
        END
```

Cholesky without Square Roots with Componentwise Augmentation - LDL^* 

(see comments for LDL^* Decomposition)

Variable	Type	Length	Contents
MR	FP	$\frac{N(N+1)}{2} + KN\ [N^2+KN]$	dividing the array into consecutive groups of length N+K, N+K-1, N+K-2, ..., 1+K [N+K], the i^{th} of these groups contains the N+1-i [N] real parts of the i^{th} row of the SCM, followed by the real parts of the i^{th} components of the K steering vectors
MI	FP	$\frac{N(N+1)}{2} + KN\ [N^2+KN]$	analogous to MR, except it contains the imaginary parts
SR			contained in MR and MI
SI			
TR	FP	N+K	temporary array
TI	FP	N+K	temporary array
RECIP	FP	N	reciprocals of elements of D in decomposition $M = LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row

Cholesky without Square Roots with Componentwise Augmentation - LDL^*

(see comments for LDL^* Decomposition) (Cont'd)

Variable	Type	Length	Contents
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LAST	I	1	location of last element of SCM

```
      BEGIN
      LAST = K*(N-1) + N*(N+1)/2 [LAST = K*(N-1) + N*N]
      STARTR = 2
      LENGTH = N+K

C     CALCULATE ROWTH ROW OF L*FACTOR IN M = LDL*

      FOR ROW = 1 TO N-1

C     *   MAKE UNIT DIAGONAL, SAVE RECIPROCAL

      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   LENGTH = LENGTH -1
      .   TR(ROW+1,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   TI(ROW+1,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROW-TH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N + K]
      .   FOR SUBROW = ROW + 1 TO N
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .               -TR(SUBROW)*MR(STARTRS,LENGTHS)
     +.   .               -TI(SUBROW)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS+1,LENGTHS-1) = MI(STARTS+1,LENGTHS-1)
     +.   .               -TR(SUBROW)*MI(STARTRS+1,LENGTHS-1)
     +.   .               +TI(SUBROW)*MR(STARTRS+1,LENGTHS-1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + K + 1]
      .   .   LENGTHS = LENGTHS -1
      .   END FOR

C     *   MOVE NORMALISED ROW FROM TR AND TI TO MR AND MI

      .   MR(STARTR,LENGTH) = TR(ROW+1,LENGTH)
      .   MI(STARTR,LENGTH) = TI(ROW+1,LENGTH)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + K + 1]
      END FOR

C     CALCULATE NTH DIAGONAL RECIPROCAL

      RECIP(N) = 1./MR(LAST)
      MR(LAST + 1,K) = RECIP(N)*MR(LAST + 1,K)
      MI(LAST + 1,K) = RECIP(N)*MI(LAST + 1,K)
      END
```

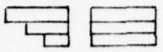
Cholesky with Square Roots with Vectorwise Augmentation - LL^*-

(see comments under LDL^* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	rowwise
MI	FP	N(N+1)/2 [N^2]	rowwise
SR	FP	K·N	real parts of K steering vectors vectorwise
SI	FP	K·N	imaginary parts of K steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition M = LL^*
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LAST	I	1	length of ROWth row starting at column ROW+1
LENGTHS	I	1	length of SUBROWth row starting in column SUBROW
AUG	I	1	current steering vector
STARTSV	I	1	location of ROW+1st element of current steering vector

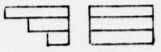
```
      BEGIN
      LAST = N*(N+1)/2 [LAST = N*N]
      STARTR = 2
      LENGTH = N

C     CALCULATE ROWTH ROW OF L* FACTOR IN M = LL*

      FOR ROW = 1 TO N-2

C     *   CALCULATE RECIPROCAL DIAGONAL AND ROW

      .   RECIP(ROW) = 1./SQRT(MR(STARTR-1))
      .   LENGTH = LENGTH-1
      .   MR(STARTR,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N]
      .   FOR SUBROW = ROW + 1 TO N-2
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .                -MR(STARTRS)*MR(STARTRS,LENGTHS)
     +.   .                -MI(STARTRS)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS+1,LENGTHS-1) = MI(STARTS+1,LENGTHS-1)
     +.   .                -MR(STARTRS)*MI(STARTRS+1,LENGTHS-1)
     +.   .                +MI(STARTRS)*MR(STARTRS+1,LENGTHS-1)
      .   .   STARTRS = STARTRS+1
      .   .   STARTS = STARTS+LENGTHS [STARTS = STARTS + N + 1]
      .   .   LENGTHS = LENGTH-1
      .   END FOR

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   MR(STARTS,2) = MR(STARTS,2) - MR(STARTRS)*
     +.              MR(STARTRS,2) - MI(STARTRS)*MI(STARTRS,2)
      .   MI(STARTS+1) = MI(STARTS+1) - MR(STARTRS)*
     +.             MI(STARTRS+1) + MI(STARTRS)*MR(STARTRS+1)
      .   STARTRS = STARTRS+1

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   MR(LAST) = MR(LAST) - MR(STARTRS)*MR(STARTRS)
     +.             -MI(STARTRS)*MI(STARTRS)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]

C     *   SUBTRACT MULTIPLES OF CURRENT ROW FROM STEERING VECTORS

      .   STARTSV = ROW + 1
      .   FOR AUG = 1 TO K
      .   .   SR(STARTSV-1) = RECIP(ROW)*SR(STARTSV-1)
      .   .   SI(STARTSV-1) = RECIP(ROW)*SI(STARTSV-1)
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .                -SR(STARTSV-1)*MR(STARTR,LENGTH)
     +.   .                -SI(STARTSV-1)*MI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .                +SR(STARTSV-1)*MI(STARTR,LENGTH)
     +.   .                -SI(STARTSV-1)*MR(STARTR,LENGTH)
      .   .   STARTSV = STARTSV + N
      .   END FOR
      END FOR
```

```
C       CALCULATE N-1ST RCV

        RECIP(N-1) = 1./SQRT(MR(STARTR-1))
        MR(STARTR) = RECIP(N-1)*MR(STARTR)
        MI(STARTR) = RECIP(N-1)*MI(STARTR)

C       CALCULATE NTH DIAGONAL RECIPROCAL

        RECIP(N) = 1./SQRT(MR(LAST) - MR(STARTR)*MR(STARTR)
     +             -MI(STARTR)*MI(STARTR))
        STARTSV = N
        FOR AUG = 1 TO K
        .   SR(STARTSV-1) = SR(STARTSV-1)*RECIP(N-1)
        .   SI(STARTSV-1) = SI(STARTSV-1)*RECIP(N-1)
        .   SR(STARTSV) = (SR(STARTSV) - SR(STARTSV-1)*MR(STARTR)
     +.                 - SI(STARTSV-1)*MI(STARTR))*RECIP(N)
        .   SI(STARTSV) = (SI(STARTSV) + SR(STARTSV-1)*MI(STARTR)
     +.                 - SI(STARTSV-1)*MR(STARTR))*RECIP(N)
        .   STARTSV=STARTSV + N
        END FOR
        END
```

Cholesky with Square Roots with Componentwise Augmentation - LL^* - 

(see comments under LL^* Decomposition)

Variable	Type	Length	Contents
MR	FP	$\frac{N(N+1)}{2} + KN\,[N^2+KN]$	dividing the array into consecutive groups of length N+K, N+K-1, N+K-2, ..., 1+K [N the i^{th} group contains the N+1-i [N] real parts of the i^{th} row of the SCM, followed by the real parts of the i^{th} components of the K steering vectors
MI	FP	$\frac{N(N+1)}{2} + KN\,[N^2+KN]$	analogous to MR, except it contains the imaginary parts
SR			contained in MR and MI
SI			
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition $M = LL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row

Cholesky with Square Roots with Componentwise Augmentation - LL^* - 

(see comments under LL^* Decomposition) (Cont'd)

Variable	Type	Length	Contents
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at column ROW+1
LENGTHS	I	1	length of SUBROWth row starting in column SUBROW

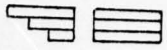
```
      BEGIN
      LAST = K*(N-1) + N*(N+1)/2 [LAST = K*(N-1) + N*N]
      STARTR = 2
      LENGTH = N + K

C     CALCULATE ROW-TH ROW OF L* FACTOR IN M = LL*

      FOR ROW = 1 TO N - 1

C     *   CALCULATE RECIPROCAL DIAGONAL AND ROW

      .   RECIP(ROW) = 1./SQRT(MR(STARTR-1))
      .   LENGTH = LENGTH - 1
      .   MR(STARTR,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROW-TH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N + K]
      .   FOR SUBROW = ROW + 1 TO N
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .                -MR(STARTRS)*MR(STARTRS,LENGTHS)
     +.   .                -MI(STARTRS)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS+1,LENGTHS-1) = MI(STARTS+1,LENGTHS-1)
     +.   .                -MR(STARTRS)*MI(STARTRS+1,LENGTHS-1)
     +.   .                +MI(STARTRS)*MR(STARTRS+1,LENGTHS-1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + K + 1]
      .   .   LENGTHS = LENGTHS - 1
      .   END FOR
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + K + 1]
      END FOR

C     CALCULATE N-TH DIAGONAL ELEMENT

      RECIP(N) = 1./SQRT(MR(LAST))
      MR(LAST+1,K) = RECIP(N)*MR(LAST+1,K)
      MI(LAST+1,K) = RECIP(N)*MI(LAST+1,K)
      END
```

Gaussian Elimination with Columnwise Identity Matrix Augmentation-GE-

Variable	Type	Length	Contents
MR	FP	N^2	rowwise
MI	FP	N^2	rowwise
SR	FP	N^2	real part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
SI	FP	N^2	imaginary part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
RECIP	FP	N	reciprocals of components of D in decomposition $M = LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st word of ROWth row
STARTS	I	1	location of ROW+1st word of SUBROWth row
LENGTH	I	1	length of ROWth row starting at column ROW+1
AUG	I	1	current column of identity matrix
STARTSV	I	1	location of ROW+1st element of current column of identity matrix
STARTSVM	I	1	location of ROWth element of current column of identity matrix

```
      BEGIN
      LENGTH = N
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO IT HAS ZEROES BELOW A UNIT DIAGONAL

      FOR ROW = 1 TO N-2
      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   LENGTH = LENGTH-1

C     *   MAKE UNIT DIAGONAL

      .   MR(STARTR,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   MI(STARTR,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   ELIMINATE BELOW THE DIAGONAL

      .   STARTS=STARTR
      .   FOR SUBROW = ROW+1 TO N
      .   .   STARTS = STARTS+N
      .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .                     -MR(STARTS-1)*MR(STARTR,LENGTH)
     +.   .                     +MI(STARTS-1)*MI(STARTR,LENGTH)
      .   .   MI(STARTS,LENGTH) = MI(STARTS,LENGTH)
     +.   .                     -MR(STARTS-1)*MI(STARTR,LENGTH)
     +.   .                     -MI(STARTS-1)*MR(STARTR,LENGTH)
      .   END FOR

C     *   ELIMINATE FROM IDENTITY MATRIX

      .   STARTSV = ROW+1
      .   FOR AUG = 1 TO ROW-1
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .                     -SR(STARTSV-1)*MR(STARTR,LENGTH)
     +.   .                     -SI(STARTSV-1)*MI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .                     +SR(STARTSV-1)*MI(STARTR,LENGTH)
     +.   .                     -SI(STARTSV-1)*MR(STARTR,LENGTH)
      .   .   STARTSV = STARTSV+N
      .   END FOR
      .   SR(STARTSV,LENGTH) = - MR(STARTR,LENGTH)
      .   SI(STARTSV,LENGTH) = MI(STARTR,LENGTH)
      .   STARTR = STARTR+N+1
      END FOR
```

```
C       ELIMINATE N-1ST COLUMN

        RECIP(N-1) = 1./MR(STARTR-1)
        MR(STARTR) = RECIP(N-1)*MR(STARTR)
        MI(STARTR) = RECIP(N-1)*MI(STARTR)

C       CALCULATE RECIPROCAL OF LAST DIAGONAL ELEMENT

        STARTS = STARTR + N
        RECIP(N) = 1./(MR(STARTS) - MR(STARTR)*MR(STARTS-1)
       +             *MI(STARTR)*MI(STARTS-1))

C       ELIMINATE FROM IDENTIFY MATRIX AND DIVIDE BY DIAGONAL ELEMENTS

        LENGTH = N
        STARTSV = N
        STARTSVM = 1
        FOR AUG = 1 TO N-2
         .    SR(STARTSV) = SR(STARTSV) - SR(STARTSV-1)
        +.                 *MR(STARTR) - SI(STARTSV-1)*MI(STARTR)
         .    SI(STARTSV) = SI(STARTSV) + SR(STARTSV-1)
        +.                 *MI(STARTR) - SI(STARTSV-1)*MR(STARTR)
         .    SR(STARTSVM,LENGTH) = RECIP(AUG,LENGTH)*SR(STARTSVM,LENGTH)
         .    SI(STARTSVM+1,LENGTH-1) = RECIP(AUG+1,LENGTH-1)*
        +.                SI(STARTSVM+1,LENGTH-1)
         .    STARTSVM = STARTSVM + N + 1
         .    STARTSV = STARTSV + N
         .    LENGTH = LENGTH - 1
         END FOR
         SR(STARTSV) = SR(STARTSV) - SR(STARTSV-1)*MR(STARTR)
        +            - SI(STARTSV-1)*MI(STARTR)
         SI(STARTSV) = SI(STARTSV) + SR(STARTSV-1)*MI(STARTR)
        +            - SI(STARTSV-1)*MR(STARTR)
         SR(STARTSVM,2) = RECIP(N-1,2)*SR(STARTSVM,2)
         SI(STARTSVM+1) = RECIP(N)*SI(STARTSVM+1)
         STARTSVM = STARTSVM + N + 1
         SR(STARTSVM) = RECIP(N)
         END
```

Gaussian Elimination with Rowwise Identity Matrix Augmentation - GE - 

Variable	Type	Length	Contents
MR	FP	$2N^2$	i^{th} group of 2N words contain real part of i^{th} row of SCM followed by real part of i^{th} row of identity matrix
MI	FP	$2N^2$	analogous to MR, but contains imaginary points
SR, SI			contained in MR and MI
RECIP	FP	N	reciprocals of elements of D in decomposition $M = LDL^*$
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st word of ROWth row
STARTS	I	1	location of ROW+1st word of SUBROWth row

```
      BEGIN
      STARTR = 2

C     ELIMINATE ROWTH COLUMN SO IT HAS ZEROES BELOW DIAGONAL

      FOR ROW = 1 TO N-1
     .    RECIP(ROW) = 1./MR(STARTR-1)

C     *    MAKE UNIT DIAGONAL

     .    MR(STARTR,N) = RECIP(ROW)*MR(STARTR,N)
     .    MI(STARTR,N-1) = RECIP(ROW)*MI(STARTR,N-1)

C     *    ELIMINATE BELOW THE DIAGONAL

     .    STARTS = STARTR
     .    FOR SUBROW = ROW+1 TO N
     .    .    STARTS = STARTS + N + N
     .    .    MR(STARTS,N) = MR(STARTS,N)
    +.    .                 -MR(STARTS-1)*MR(STARTR,N)
    +.    .                 +MI(STARTS-1)*MI(STARTR,N)
     .    .    MI(STARTS,N) = MI(STARTS,N)
    +.    .                 -MR(STARTS-1)*MI(STARTR,N)
    +.    .                 -MI(STARTS-1)*MR(STARTR,N)
     .    END FOR
     .    STARTR = STARTR + N + N + 1
      END FOR
      RECIP(N) = 1./MR(STARTR-1)
      MR(STARTR,N) = RECIP(N)*MR(STARTR,N)
      MI(STARTR,N-1) = RECIP(N)*MI(STARTR,N-1)
      END
```

Cholesky without Square Roots with Columnwise Identity Matrix Augmentation - LDL*-

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	rowwise
MI	FP	$N(N+1)/2$ [N^2]	rowwise
SR	FP	N^2	real part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
SI	FP	N^2	imaginary part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
TR	FP	N	temporary array
TI	FP	N	temporary array
RECIP	FP	N	reciprocals of elements of D in decomposition M = LDL*
TR1	FP	1	temporary location
TI1	FP	1	temporary location
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row

Cholesky without Square Roots with Columnwise Identity Matrix Augmentation (Cont'd) -LDL*-

Variable	Type	Length	Contents
STARTRS	I	1	location of SUBROWth elements of ROWth row
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at element ROW+1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW
AUG	I	1	current column of identity matrix
STARTSV	I	1	location of ROW+1st element of current column of identity matrix
STARTSVM	I	1	location of ROWth element of current column of identity matrix

```
      BEGIN
      LAST = N*(N+1)/2 [LAST = N*N]
      STARTR = 2
      LENGTH = N

C     CALCULATE ROWTH ROW OF L* FACTOR IN H = LDL*

      FOR ROW = 1 TO N-2

C     *   MAKE UNIT DIAGONAL, SAVE RECIPROCAL

      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   LENGTH = LENGTH-1
      .   TR(ROW+1,LENGTH) = RECIP(ROW)*MR(STARTR,LENGTH)
      .   TI(ROW+1,LENGTH) = RECIP(ROW)*MI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR+N]
      .   FOR SUBROW = ROW+1 TO N-2
      .   .   MR(STARTS,LENGTHS) = MR(STARTS,LENGTHS)
     +.   .               -TR(SUBROW)*MR(STARTRS,LENGTHS)
     +.   .               -TI(SUBROW)*MI(STARTRS,LENGTHS)
      .   .   MI(STARTS+1,LENGTHS-1) = MI(STARTS+1,LENGTHS-1)
     +.   .               -TR(SUBROW)*MI(STARTS+1,LENGTHS-1)
     +.   .               +TI(SUBROW)*MR(STARTS+1,LENGTHS-1)
      .   .   STARTRS = STARTRS+1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS+N+1]
      .   .   LENGTHS = LENGTHS -1
      .   END FOR

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   MR(STARTS,2) = MR(STARTS,2) -TR(N-1)*MR(STARTRS,2)
     +.               -TI(N-1)*MI(STARTRS,2)
      .   MI(STARTS+1) = MI(STARTS+1) -TR(N-1)*MI(STARTRS+1)
     +.               +TI(N-1)*MR(STARTRS+1)
      .   STARTRS = STARTRS+1

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   MR(LAST) = MR(LAST) -TR(N)*MR(STARTRS)
     +.              -TI(N)*MI(STARTRS)

C     *   MOVE NORMALIZED ROW FROM TR AND TI TO MR AND MI

      .   MR(STARTR,LENGTH) = TR(ROW+1,LENGTH)
      .   MI(STARTR,LENGTH) = TI(ROW+1,LENGTH)
      .   STARTR = STARTR+LENGTH+1 [STARTR = STARTR + N + 1]

C     *   ELIMINATE FROM IDENTITY MATRIX

      .   STARTSV = ROW+1
      .   FOR AUG = 1 TO ROW-1
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .               -SR(STARTSV-1)*TR(ROW+1,LENGTH)
     +.   .               -SI(STARTSV-1)*TI(ROW+1,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .               +SR(STARTSV-1)*TI(ROW+1,LENGTH)
     +.   .               -SI(STARTSV-1)*TR(ROW+1,LENGTH)
      .   .   STARTSV = STARTSV + N
      .   END FOR
      .   SR(STARTSV,LENGTH) = - MR(STARTR,LENGTH)
      .   SI(STARTSV,LENGTH) = MI(STARTR,LENGTH)
      END FOR
```

```
C     CALCULATE N-1ST ROW

      RECIP(N-1) = 1./MR(STARTR-1)
      TR1 = RECIP(N-1)*MR(STARTR)
      TR2 = RECIP(N-1)*MI(STARTR)

C     CALCULATE NTH DIAGONAL RECIPROCAL

      RECIP(N) = 1./(MR(LAST) -TR1*MR(STARTR)
     +           -TI1*MI(STARTR))
      MR(STARTR) = TR1
      MI(STARTR) = TI1

C     ELIMINATE FROM IDENTITY MATRIX AND DIVIDE BY DIAGONAL ELEMENTS

      LENGTH = N
      STARTSV = N
      STARTSVM = 1
      FOR AUG = 1 TO N-2
       .   SR(STARTSV) = SR(STARTSV) -SR(STARTSV-1)*TR1
      +.                 -SI(STARTSV-1)*TI1
       .   SI(STARTSV) = SI(STARTSV) + SR(STARTSV-1)*TI1
      +.                 -SI(STARTSV-1)*TR1
       .   SR(STARTSVM,LENGTH) = RECIP(AUG,LENGTH)*SR(STARTSVM,LENGTH)
       .   SI(STARTSVM+1,LENGTH-1) = RECIP(AUG+1,LENGTH-1)*
      +.                 SI(STARTSVM+1,LENGTH-1)
       .   STARTSVM = STARTSVM + N + 1
       .   STARTSV = STARTSV + 1
       .   LENGTH = LENGTH -1
       END FOR
       SR(STARTSV) = SR(STARTSV) -SR(STARTSV-1)*TR1-SI(STARTSV-1)*TI1
       SI(STARTSV) = SI(STARTSV) + SR(STARTSV-1)*TI1
      +            -SI(STARTSV-1)*TR1
       SR(STARTSVM,2) = RECIP(N-1,2)*SR(STARTSVM,2)
       SI(STARTSVM+1) = RECIP(N)*SI(STARTSVM+1)
       STARTSVM = STARTSVM + N + 1
       SR(STARTSVM) = RECIP(N)
       END
```

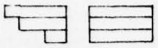
Cholesky without Square Roots with Rowwise Identity Matrix Augmentation

- LDL*-

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$\frac{N(N+1)}{2} + N^2$ [$2N^2$]	dividing the array into consecutive groups of length N+K, N+K-1, N+K-2,...,1+K[N+K], the i^{th} of these groups contains the N+1-i[N] real parts of the i^{th} row of the SCM, followed by the real part of the i^{th} row of the identity matrix
MI	FP	$\frac{N(N+1)}{2} + N^2$ [$2N^2$]	analogous to MR, except it contains the imaginary parts
SR			} contained in MR and MI
SI			
TR	FP	N	temporary array
TI	FP	N	temporary array
RECIP	FP	N	reciprocals of elements of D in decomposition M = LDL*
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row

Cholesky without Square Roots with Rowwise Identity Matrix Augmentation (Cont'd)

- LDL* - 

Variable	Type	Length	Contents
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of SUBROWth row starting at element SUBROW

```
      BEGIN
      LAST = N*(N-1) + N + (N+1)/2 [LAST = N*(2*N-1)]
      STARTR = 2

C     CALCULATE ROWTH ROW OF L* FACTOR IN M = LDL*

      FOR ROW = 1 TO N-1

C     *   MAKE UNIT DIAGONAL, SAVE RECIPROCAL

      .   RECIP(ROW) = 1./MR(STARTR-1)
      .   TR(ROW+1,N) = RECIP(ROW)*MR(STARTR,N)
      .   TI(ROW+1,N-1) = RECIP(ROW)*MI(STARTR,N-1)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTH = N
      .   STARTRS = START
      .   STARTS = STARTR + N + N-ROW [STARTS = STARTR + N + N]
      .   FOR SUBROW = ROW + 1 TO N
      .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .                -TR(SUBROW)*MR(STARTRS,LENGTH)
     +.   .                -TI(SUBROW)*MI(STARTRS,LENGTH)
      .   .   MI(STARTS+1,LENGTH-1) = MI(STARTS+1,LENGTH-1)
     +.   .                -TR(SUBROW)*MI(STARTRS+1,LENGTH-1)
     +.   .                +TI(SUBROW)*MR(STARTRS+1,LENGTH-1)
      .   .   STARTRS = STARTRS + 1
      .   .   STARTS = STARTS + N + N+1 - SUBROW [STARTS = STARTS + N + N + 1]
      .   .   LENGTH = LENGTH-1
      .   END FOR

C     *   MOVE NORMALIZED ROW FROM TR AND TI TO MR AND MI

      .   MR(STARTR,N) = TR(ROW+1,N)
      .   MI(STARTR,N-1) = TR(ROW + 1,N-1)
      .   STARTR = STARTR + N + N+1-ROW [STARTR = STARTR + N + N + 1]
      END FOR

C     CALCULATE NTH DIAGONAL RECIPROCAL

      RECIP(1) = 1./MR(LAST)
      MR(LAST+1,N) = RECIP(N)*MR(LAST+1,N)
      MI(LAST+1,N-1) = RECIP(N)*MI(LAST+1,N-1)
      END
```

Cholesky with Square Roots with Columnwise Identity Matrix Augmentation

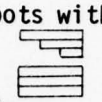
- LL* - 

(see comments under LL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	rowwise
MI	FP	$N(N+1)/2$ [N^2]	rowwise
SR	FP	N^2	real part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
SI	FP	N^2	imaginary part of identity matrix columnwise (i.e., not interleaved with rows of SCM)
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition M = LL*
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth elements of ROWth row
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at element ROW+1

Cholesky with Square Roots with Columnwise Identity Matrix Augmentation (Cont'd)

- LL* -

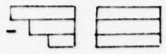


Variable	Type	Length	Contents
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW
AUG	I	1	current column of identity matrix
STARTSV	I	1	location of ROW+1st element of current column of identity matrix

```
      BEGIN
      LAST = N*(N+1)/2 [LAST = N*N]
      STARTR = 2
      LENGTH = N

C     CALCULATE ROWTH ROW OF L* FACTOR IN H = LL*

      FOR ROW = 1 TO N-2

C     *   CALCULATE RECIPROCAL DIAGONAL AND ROW

      .   RECIP(ROW) = 1./SQRT(HR(STARTR-1))
      .   LENGTH = LENGTH-1
      .   HR(STARTR,LENGTH) = RECIP(ROW)*HR(STARTR,LENGTH)
      .   HI(STARTR,LENGTH) = RECIP(ROW)*HI(STARTR,LENGTH)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTHS = LENGTH
      .   STARTRS = STARTR
      .   STARTS = STARTR + LENGTH [STARTS = STARTR + N]
      .   FOR SUBROW = ROW+1 TO N-2
      .   .   HR(STARTS,LENGTHS) = HR(STARTS,LENGTHS)
     +.   .                -HR(STARTRS)*HR(STARTRS,LENGTHS)
     +.   .                -HI(STARTRS)*HI(STARTRS,LENGTHS)
      .   .   HI(STARTS+1,LENGTHS-1) = HI(STARTS+1,LENGTHS)
     +.   .                -HR(STARTRS)*HI(STARTRS+1,LENGTHS-1)
     +.   .                +HI(STARTRS)*HR(STARTRS+1,LENGTHS-1)
      .   .   STARTRS = STARTRS+1
      .   .   STARTS = STARTS + LENGTHS [STARTS = STARTS + N + 1]
      .   .   LENGTHS = LENGTH -1
      .   END FOR

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM N-1ST ROW

      .   HR(STARTS,2) = HR(STARTS,2) -HR(STARTRS)*
     +.                  HR(STARTRS,2) -HI(STARTRS)*HI(STARTRS,2)
      .   HI(STARTS+1) = HI(STARTS + 1) -HR(STARTRS)*
     +.                  HI(STARTRS+1) +HI(STARTRS)*HR(STARTRS+1)
      .   STARTRS = STARTRS + 1

C     *   SUBTRACT MULTIPLE OF ROWTH ROW FROM NTH ROW

      .   HR(LAST) = HR(LAST) -HR(STARTRS)*HR(STARTRS)
     +.                -HI(STARTRS)*HI(STARTRS)
      .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]

C     *   ELIMINATE IDENTITY MATRIX

      .   STARTSV = ROW+1
      .   FOR NC = 1 TO ROW-1
      .   .   SR(STARTSV-1) = RECIP(ROW)*SR(STARTSV-1)
      .   .   SI(STARTSV-1) = RECIP(ROW)*SI(STARTSV-1)
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .                -SR(STARTSV-1)*HR(STARTR,LENGTH)
     +.   .                +SI(STARTSV-1)*HI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .                -SR(STARTSV-1)*HI(STARTR,LENGTH)
     +.   .                -SI(STARTSV-1)*HR(STARTR,LENGTH)
      .   .   STARTSV = STARTSV + N
      .   END FOR
      .   SR(STARTSV-1) = RECIP(LAST)
      .   SR(STARTSV,LENGTH) = -HR(STARTR,LENGTH)*RECIP(ROW)
      .   SI(STARTSV,LENGTH) = HI(STARTR,LENGTH)
      END FOR
```

```
C       CALCULATE N-1ST ROW

        RECIP(N-1) = 1./SQRT(MR(STARTR-1))
        MR(STARTR) = RECIP(N-1)*MR(STARTR)
        MI(STARTR) = RECIP(N-1)*MI(STARTR)

C       CALCULATE NTH DIAGONAL RECIPROCAL

        RECIP(N) = 1./SQRT(MR(LAST) - MR(STARTR)*MR(STARTR)
       +           -MI(STARTR)*MI(STARTR))
        STARTSV=N
        FOR AUG=1 TO N-1
        .   SR(STARTSV-1) = RECIP(N-1)*SR(STARTSV-1)
        .   SI(STARTSV-1) = RECIP(N-1)*SI(STARTSV-1)
        .   SR(STARTSV) = (SR(STARTSV) -SR(STARTSV-1)*
       +.             MR(STARTR) -SI(STARTSV-1)*MI(STARTR))*RECIP(N)
        .   SI(STARTSV) = (SI(STARTSV) + SR(STARTSV-1))*
       +.             MI(STARTR) -SI(STARTSV-1)*MR(STARTR))
       *.              *RECIP(N)
        .   STARTSV= STARTSV+N
        END FOR
        SR(STARTSV) = RECIP(N)
        END
```

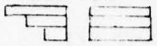
Cholesky with Square Roots with Rowwise Identity Matrix Augmentation

- LL* - 

(see comments under LL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$\frac{N(N+1)}{2} + N^2$ [$2N^2$]	dividing the array into consecutive groups of length N+K, N+K-1, N+K-2,...,1+K[N+K], the i^{th} of these groups contains the N+1-i[N] real parts of the i^{th} row of the SCM, followed by the real part of the i^{th} row of the identity matrix
MI	FP	$\frac{N(N+1)}{2} + N^2$ [$2N^2$]	analogous to MR, except it contains the imaginary parts
SR			contained in MR and MI
SI			contained in MR and MI
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition M = LL*
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROW+1st element of ROWth row
STARTS	I	1	location of SUBROWth element of SUBROWth row

Cholesky with Square Roots with Rowwise Identity Matrix Augmentation (Cont'd)



- LL* -

Variable	Type	Length	Contents
STARTRS	I	1	location of SUBROWth elements of ROWth row
LAST	I	1	location of last element of SCM
LENGTH	I	1	length of ROWth row starting at element ROW+1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW

```
      BEGIN
      LAST = N*(N-1) + N*(N+1)/2 [LAST = N*(2*N-1)]
      STARTR = 2

C     CALCULATE NTH ROW OF L* FACTOR IN M = LL*

      FOR ROW = 1 TO N-1

C     *   CALCULATE RECIPROCAL DIAGONAL AND ROW

      .   RECIP(ROW) = 1./SQRT(MR(STARTR-1))
      .   MR(STARTR,N) = RECIP(ROW)*MR(STARTR,N)
      .   MI(STARTR,N-1) = RECIP(ROW)*MI(STARTR,N-1)

C     *   SUBTRACT MULTIPLES OF ROWTH ROW FROM OTHER ROWS

      .   LENGTH = N
      .   STARTRS = STARTR
      .   STARTS = STARTR + N + N-ROW [STARTS = STARTR + N + N]
      .   FOR SUBROW = ROW + 1 TO N
      .   .   MR(STARTS,LENGTH) = MR(STARTS,LENGTH)
     +.   .                -MR(STARTRS)*MR(STARTRS,LENGTH)
     +.   .                -MI(STARTRS)*MI(STARTRS,LENGTH)
      .   .   MI(STARTS+1,LENGTH-1) = MI(STARTS+1,LENGTH-1)
     +.   .                -MR(STARTRS)*MI(STARTRS+1,LENGTH-1)
     +.   .                +MI(STARTRS)*MR(STARTRS+1,LENGTH-1)
      .   .   STARTRS = STARTRS +1
      .   .   STARTS = STARTS + N + N + 1-SUBROW [STARTS = STARTS + N + N + 1]
      .   .   LENGTH = LENGTH -1
      .   END FOR
      .   STARTRS = STARTR + N + N + 1-ROW [STARTR = STARTR + N + N + 1]
      END FOR

C     CALCULATE NTH DIAGONAL ELEMENT

      RECIP(1) = 1./MR(LAST)
      MR(LAST + 1,N) = RECIP(N)*MR(LAST+1,N)
      MI(LAST+1,N-1) = RECIP(1)*MI(LAST+1,N-1)
      END
```

First Back Substitution for LDL* or GE with L* Stored Rowwise and $\overline{S}$ Vectorwise

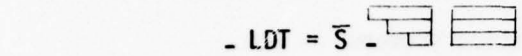
- LDT = $\overline{S}$ - 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	real part of L* factor rowwise
MI	FP	N(N+1)/2 [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of elements of D factors
AUG	I	1	current steering vector
ROW	I	1	current row
STARTSV	I	1	location of first element of current steering vector
STARTSVA	I	1	location of ROW+1st element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1
STARTR	I	1	location of ROW+1st element of ROWth row of SCM

```
      BEGIN
      STARTSV=1

C     DO EACH STEERING VECTOR

      FOR AUG = 1 TO K

C     *       SUBTRACT MULTIPLES OF EACH ROW OF M FROM THE STEERING VECTORS

       .    STARTSVA = STARTSV + 1
       .    STARTR = 2
       .    LENGTH = N-1
       .    FOR ROW = 1 TO N-2
       .    .    SR(STARTSVA,LENGTH) = SR(STARTSVA,LENGTH)
      +.    .                -SR(STARTSVA-1)*MR(STARTR,LENGTH)
      +.    .                -SI(STARTSVA-1)*MI(STARTR,LENGTH)
       .    .    SI(STARTSVA,LENGTH) = SI(STARTSVA,LENGTH)
      +.    .                +SR(STARTSVA-1)*MI(STARTR,LENGTH)
      +.    .                -SI(STARTSVA-1)*MR(STARTR,LENGTH)
       .    .    STARTSVA = STARTSVA + 1
       .    .    STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]
       .    .    LENGTH = LENGTH-1
       .    END FOR
       .    SR(STARTSVA) = SR(STARTSVA) -SR(STARTSVA-1)*
      +.               MR(STARTR) -SI(STARTSVA-1)*MI(STARTR)
       .    SI(STARTSVA) = SI(STARTSVA) + SR(STARTSVA-1)*
      +.               MI(STARTR) -SI(STARTSVA-1)*MR(STARTR)

C     *       MULTIPLY BY RECIPROCAL DIAGONALS

       .    SR(STARTSV,N) = RECIP(1,N) * SR(STARTSV,N)
       .    SI(STARTSV,N) = RECIP(1,N) * SI(STARTSV,N)
       .    STARTSV = STARTSV + N
      END FOR
      END
```

First Back Substitution for LDL* or GE with L* Stored Rowwise and Componentwise

$-LD\Gamma = \bar{S}$ 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* factor rowwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors componentwise
SI	FP	KN	imaginary part of steering vectors componentwise
RECIP	FP	N	reciprocals of elements of D
AUG	I	1	current steering vector
ROW	I	1	current row
STARTSV	I	1	location of beginning of ROWth components of steering vectors
SUBROW	I	1	row from which multiple of current row is to be subtracted
STARTSVS	I	1	location of beginning of SUBROWth components of steering vectors
STARTR	I	1	location of the (ROW, SUBROW)th element of the SCM

```
      BEGIN

C     SUBTRACT MULTIPLES OF ROWTH COMPONENTS OF STEERING VECTORS

      FROM SUBROWTH COMPONENTS
      STARTSV = 1
      STARTR = 2
      FOR ROW = 1 TO N-1
      .   STARTSVS = STARTSV+K
      .   FOR SUBROW = ROW + 1 TO N
      .   .   SR(STARTSVS,K) = SR(STARTSVS,K)
     +.   .                  -MR(STARTR)*SR(STARTSV,K)
     +.   .                  -MI(STARTR)*SI(STARTSV,K)
      .   .   SI(STARTSVS,K) = SI(STARTSVS,K)
     +.   .                  +MI(STARTR)*SR(STARTSV,K)
     +.   .                  -MR(STARTR)*SI(STARTSV,K)
      .   .   STARTR = STARTR + 1
      .   .   STARTSVS = STARTSVS + K
      .   END FOR
      .   SR(STARTSV,K) = RECIP(ROW)*SR(STARTSV,K)
      .   SI(STARTSV,K) = RECIP(ROW)*SI(STARTSV,K)
      .   STARTSV = STARTSV + K
      .   [STARTR = STARTR + ROW]
      END FOR
      SR(STARTSV,K) = RECIP(N)*SR(STARTSV,K)
      SI(STARTSV,K) = RECIP(N)*SI(STARTSV,K)
      END
      (NOTE THAT ALL VECTOR OPERATIONS ARE OF LENGTH
      K AND SO TO USE THIS ALGORITHM K SHOULD
      BE GREATER THAN 1)
```

First Back Substitution for LDL* or GE with L* Stored Columnwise and $\overline{S}$ Vectorwise

$- LDT = \overline{S} -$ 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* stored componentwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* stored componentwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of elements of D
T	FP	2N	temporary array used with sum
DOT	FP	1	temporary location
AUG	I	1	current steering vector
ROW	I	1	current column of SCM
STARTSV	I	1	location of first element of current steering vector
STARTR	I	1	location of beginning of current column of the SCM
STARTSVS	I	1	location of current element of steering vector being calculated

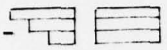
```
      BEGIN
      STARTSV = 1

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO K
      .   STARTR = 2

C     *       ELIMINATE THE 2ND ELEMENT OF THE STEERING VECTOR

      .   SR(STARTSV+1) = SR(STARTSV+1) -MR(STARTR)*
     +.                   SR(STARTSV) -MI(STARTR)*SI(STARTSV)
      .   SI(STARTSV+1) = SI(STARTSV+1) -MR(STARTR)*
     +.                   SI(STARTSV) +MI(STARTR)*SR(STARTSV)

C     *       ELIMINATE THE 3RD THROUGH NTH ELEMENTS OF THE STEERING VECTORS

      .   STARTSVS = STARTSV+2
      .   FOR ROW = 3 TO N
      .   .   T(1,ROW-1) = SR(STARTSV, ROW-1)*MR(STARTR,ROW-1)
      .   .   T(ROW,ROW-1) = SI(STARTSV,ROW-1)*MI(STARTR,ROW-1)
      .   .   DOT = SUM(T(1,2*ROW-2))
      .   .   SR(STARTSVS) = SR(STARTSVS) -DOT
      .   .   T(1,ROW-1) = - SR(STARTSV,ROW-1)*MI(STARTR,ROW-1)
      .   .   T(ROW,ROW-1) = SI(STARTSV,ROW-1)*MR(STARTR,ROW-1)
      .   .   DOT = SUM(T(1,2*ROW-2))
      .   .   SI(STARTSVS) = SI(STARTSVS) -DOT
      .   .   STARTSVS = STARTSVS+1
      .   .   STARTR = STARTR+ROW [STARTR = STARTR+N]
      .   END FOR
      .   SR(STARTSV,N) = SR(STARTSV,N)*RECIP(1,N)
      .   SI(STARTSV,N) = SI(STARTSV,N)*RECIP(1,N)
      .   STARTSV = STARTSV+N
      END FOR
      END
```

First Back Substitution for LDL* or GE with L* Stored Columnwise and $\overline{S}$ Componentwise

$-LDT = \overline{S} -$ 

(This method is identical to the First Back Substitution with L* stored rowwise and $\overline{S}$ componentwise, except in the calculation of subscripts of MR and MI, which do not change the operation counts.)

First Back Substitution for LL* with L* Stored Rowwise and $\bar{S}$ Vectorwise

- LT = $\bar{S}$ - 

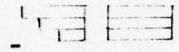
(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	real part of L* factor rowwise
MI	FP	N(N+1)/2 [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L factors
AUG	I	1	current steering vector
ROW	I	1	current row
STARTSV	I	1	location of first element of current steering vector
STARTSVA	I	1	location of ROW+1st element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1
STARTR	I	1	location of ROW+1st element of ROWth row of SCM

```
      BEGIN
      STARTSV = 1

C     DO EACH STEERING VECTOR

      FOR AUG = 1 TO K

C     *       SUBTRACT MULTIPLES OF EACH ROW OF M FROM THE STEERING VECTORS

      .   STARTSVA = STARTSV+1
      .   STARTR = 2
      .   LENGTH = N-1
      .   FOR ROW = 1 TO N-2
      .   .   SR(STARTSVA-1) = RECIP(ROW)*SR(STARTSVA-1)
      .   .   SI(STARTSVA-1) = RECIP(ROW)*SI(STARTSVA-1)
      .   .   SR(STARTSVA,LENGTH) = SR(STARTSVA,LENGTH)
     +.   .                -SR(STARTSVA-1)*MR(STARTR,LENGTH)
     +.   .                -SI(STARTSVA-1)*MI(STARTR,LENGTH)
      .   .   SI(STARTSVA,LENGTH) = SI(STARTSVA,LENGTH)
     +.   .                +SR(STARTSVA-1)*MI(STARTR,LENGTH)
     +.   .                -SI(STARTSVA-1)*MR(STARTR,LENGTH)
      .   .   STARTSVA = STARTSVA+1
      .   .   STARTR = STARTR + LENGTH + 1 [STARTR = STARTR + N + 1]
      .   .   LENGTH = LENGTH-1
      .   END FOR
      .   SR(STARTSVA-1) = RECIP(N-1)*SR(STARTSVA-1)
      .   SI(STARTSVA-1) = RECIP(N-1)*SI(STARTSVA-1)
      .   SR(STARTSVA) = SR(STARTSVA) -SR(STARTSVA-1)*
     +.              MR(STARTR) - SI(STARTSVA-1)*MI(STARTR)
      .   SI(STARTSVA) = SI(STARTSVA) + SR(STARTSVA-1)*
     +.              MI(STARTR) - SI(STARTSVA-1)*MR(STARTR)
      .   SR(STARTSVA) = RECIP(N)*SR(STARTSVA)
      .   SI(STARTSVA) = RECIP(N)*SI(STARTSVA)
      .   STARTSV = STARTSV + N
      END FOR
      END
```

First Back Substitution for LL* with L* Stored Rowwise and $\overline{S}$ Componentwise

$-LT = \overline{S}$ -



(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	real part of L* factor rowwise
MI	FP	N(N+1)/2 [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors componentwise
SI	FP	KN	imaginary part of steering vectors componentwise
RECIP	FP	N	reciprocals of diagonal elements of L factor
AUG	I	1	current steering vector
ROW	I	1	current row
STARTSV	I	1	location of beginning of ROWth components of steering vectors row from which multiple of current row is to be subtracted
STARTSVS	I	1	location of beginning of SUBROWth components of steering vectors
STARTR	I	n	location of the (ROW, SUBROW)th element of the SCM

```
      BEGIN

C     SUBTRACT MULTIPLES OF ROWTH COMPONENTS OF STEERING

      VECTORS FROM SUBROWTH COMPONENTS
      STARTSV = 1
      STARTR = 2
      FOR ROW = 1 TO N-1
      .   STARTSVS = STARTSV+K
      .   SR(STARTSV,K) = RECIP(ROW)*SR(STARTSV,K)
      .   SI(STARTSV,K) = RECIP(ROW)*SI(STARTSV,K)
      .   FOR SUBROW = ROW+1 TO N
      .   .   SR(STARTSVS,K) = SR(STARTSVS,K)
     +.   .                  -NR(STARTR)*SR(STARTSV,K)
     +.   .                  -NI(STARTR)*SI(STARTSV,K)
      .   .   SI(STARTSVS,K) = SI(STARTSVS,K)
     +.   .                  +NI(STARTR)*SR(STARTSV,K)
     +.   .                  -NR(STARTR)*SI(STARTSV,K)
      .   .   STARTR = STARTR + 1
      .   .   STARTSVS = STARTSVS + K
      .   END FOR
      .   STARTSV = STARTSV + K
      .   [STARTR = STARTR + ROW]
      END FOR
      SR(STARTSV,K) = RECIP(N)*SR(STARTSV,K)
      SI(STARTSV,K) = RECIP(N)*SI(STARTSV,K)
      END
      (NOTE THAT ALL VECTOR OPERATIONS ARE OF
      LENGTH K AND SO TO USE THIS ALGORITHM
      K SHOULD BE GREATER THAN 1)
```

First Back Substitution for LL* with L* Stored Columnwise and $\overline{S}$ Vectorwise

- LT = $\overline{S}$ - 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	real part of L* stored componentwise
MI	FP	N(N+1)/2 [N^2]	imaginary part of L* stored componentwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of elements of D
T	FP	2N	temporary array used with sum
DOT	FP	1	temporary location
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location of first element of current steering vector
STARTR	I	1	location of beinning of current row of the SCM
STARTSVS	I	1	location of current element of steering vector being calculated

```
      BEGIN
      STARTSV = 1

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO K
      .   STARTR = 2
      .   SR(STARTSV) = SR(STARTSV)*RECIP(1)
      .   SI(STARTSV) = SI(STARTSV)*RECIP(1)

C     *    ELIMINATE THE 2ND ELEMENT OF THE STEERING VECTOR

      .   SR(STARTSV+1) = SR(STARTSV+1) - MR(STARTR)*
     +.                   SR(STARTSV) - MI(STARTR)*SI(STARTSV)
      .   SI(STARTSV+1) = SI(STARTSV+1) - MR(STARTR)*
     +.                   SI(STARTSV) + MI(STARTR)*SR(STARTSV)
      .   SR(STARTSV+1) = SR(STARTSV+1)*RECIP(2)
      .   SI(STARTSV+1) = SI(STARTSV+1)*RECIP(2)

C     *    ELIMINATE THE 3RD THROUGH NTH ELEMENTS OF THE STEERING VECTORS

      .   STARTSVS = STARTSV+2
      .   FOR ROW = 3 TO N
      .   .   T(1,ROW-1) = SR(STARTSV,ROW-1)*MR(STARTR,ROW-1)
      .   .   T(ROW,ROW-1) = SI(STARTSV,ROW-1)*MI(STARTR,ROW-1)
      .   .   DOT = SUM(T(1,2*ROW-2))
      .   .   SR(STARTSVS) = SR(STARTSVS) - DOT
      .   .   T(1,ROW-1) = -SR(STARTSV,ROW-1)*MI(STARTR,ROW-1)
      .   .   T(ROW,ROW-1) = SI(STARTSV,ROW-1)*MR(STARTR,ROW-1)
      .   .   DOT = SUM(T(1,2*ROW-2))
      .   .   SI(STARTSVS) = SI(STARTSVS) - DOT
      .   .   STARTSVS = STARTSVS+1
      .   .   STARTR = STARTR*ROW [STARTR = STARTR + 1]
      .   .   SR(STARTSVS) = RECIP(ROW)*SR(STARTSVS)
      .   .   SI(STARTSVS) = RECIP(ROW)*SI(STARTSVS)
      .   END FOR
      .   STARTSV = STARTSV + N
      END FOR
      END
```

First Back Substitution for LL* with L* Stored Columnwise and $\overline{S}$ Componentwise

- LT = S- 

(This method is identical to First Back Substitution with L* stored rowwise and $\overline{S}$ componentwise, except in the calculation of subscripts of MR and MI, which does not change the operation counts.)

Second Back Substitution for LDL* or GE with L* Stored Rowwise and $\bar{S}$ Vectorwise

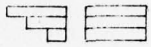
$-L^*W = T-$ 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* factor rowwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
T	FP	2N	temporary array used with sum
DOT	FP	1	temporary location
LAST	I	1	location of last element of SCM
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location of last element of current steering vector
STARTR	I	1	location of ROW+1st element of ROWth row of the SCM
STARTSVS	I	1	location of ROW+1st element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1

```
      BEGIN
      STARTSV = N
      LAST = N*(N+1)/2 [LAST = N*N]

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO K
      .   STARTR = LAST -1 [STARTR = LAST -N]

C     *   CALCULATE N-1ST COMPONENT

      .   SR(STARTSV -1) = SR(STARTSV-1) -MR(STARTR)*
     +.                   SR(STARTSV) + MI(STARTR)*SI(STARTSV)
      .   SI(STARTSV-1) = SI(STARTSV-1) - MR(STARTR)*
     +.                   SI(STARTSV) - MI(STARTR)*SR(STARTSV)
      .   STARTSVS = STARTSV-1
      .   LENGTH = 2
      .   FOR ROW = N-2 TO 1 STEP -1
      .   .   STARTR = STARTR - LENGTH -1 [STARTR = STARTR -N -1]
      .   .   T(1,LENGTH) = MR(STARTR,LENGTH)*
     +.   .                 SR(STARTSVS,LENGTH)
      .   .   T(LENGTH+1,LENGTH) -MI(STARTR,LENGTH)
     +.   .                 *SI(STARTSVS,LENGTH)
      .   .   DOT = SUM(T(1,2*LENGTH))
      .   .   SR(STARTSVS-1) = SR(STARTSVS-1) - DOT
      .   .   T(1,LENGTH) = MR(STARTR,LENGTH)*SI(STARTSVS,LENGTH)
      .   .   T(LENGTH+1, LENGTH) = MI(STARTR,LENGTH)*SR(STARTSVS,LENGTH)
      .   .   DOT = SUM (T(1,2*LENGTH))
      .   .   SI(STARTSVS-1) = SI(STARTSVS-1) - DOT
      .   .   STARTSVS = STARTSVS -1
      .   .   LENGTH = LENGTH + 1
      .   END FOR
      .   STARTSV = STARTSV + N
      END FOR
      END
```

Second Back Substitution for LDL* or GE with L* Stored Rowwise and $\overline{S}$ Componentwise

- L*W = T - 

(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2 [N^2]	real part of L* factor rowwise
MI	FP	N(N+1)/2 [N^2]	imaginary part of L* factor rowwise
SR	FP	KN	real part of K steering vectors componentwise
SI	FP	KN	imaginary parts of K steering vectors componentwise
ROW	I	1	current row of SCM
STARTSV	I	1	beginning location of ROWth components of the steering vectors
SUBROW	I	1	row of components from which multiple of ROWth row is subtracted
STARTR	I	1	location of (SUBROW, ROW)th element of the SCM
STARTSVS	I	1	beginning location of SUBROWth components of the steering vectors

```
      BEGIN
      STARTSV = K*(N-1) +1

C     SUBTRACT MULTIPLES OF EACH COMPONENT FROM PREVIOUS COMPONENTS

      FOR ROW = N TO 2 STEP -1
      .   STARTR = ROW
      .   STARTSVS = 1
      .   FOR SUBROW = 1 TO ROW -1
      .   .   SR(STARTSVS,K) = SR(STARTSVS,K)
     +.   .                  -MR(STARTR)*SR(STARTSV,K)
     +.   .                  +MI(STARTR)*SI(STARTSV,K)
      .   .   SI(STARTSVS,K) = SI(STARTSVS,K)
     +.   .                  -MR(STARTR)*SI(STARTSV,K)
     +.   .                  -MI(STARTR)*SR(STARTSV,K)
      .   .   STARTR = STARTR + N -SUBROW [STARTR = STARTR + N]
      .   .   STARTSVS = STARTSVS + K
      .   END FOR
      .   STARTSV = STARTSV -K
      END FOR
      END
      (NOTE K SHOULD BE GREATER THAN 1)
```

Second Back Substitution for LDL* or GE with L* Stored Columnwise and $\bar{S}$ Vectorwise

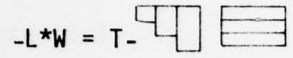
(see comments under LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* columnwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* columnwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
LAST	I	1	location of the last element of SCM
AUG	I	1	current steering vector
STARTSV	I	1	location of first element of current steering vector
ROW	I	1	current column of SCM
STARTR	I	1	beginning location of current column of SCM
STARTVS	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of current column of SCM excluding diagonal element

```
      BEGIN

C     ELIMINATE EACH STEERING VECTOR

      STARTSV = 1
      LAST = N*(N+1)/2 [LAST = N*N]
      FOR AUG = 1 TO K
      .   STARTSVS = STARTSV + K -1
      .   STARTR = LAST - N + 1
      .   LENGTH = N -1
      .   FOR ROW = N TO 3 STEP -1
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .                -SR(STARTSVS)*MR(STARTR,LENGTH)
     +.   .                +SI(STARTSVS)*MI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .                -SR(STARTSVS)*MI(STARTR,LENGTH)
     +.   .                -SI(STARTSVS)*MR(STARTR,LENGTH)
      .   .   STARTSVS = STARTSVS-1
      .   .   STARTR = STARTR - LENGTH
      .   .   LENGTH = LENGTH -1
      .   END FOR
      .   SR(STARTSV) = SR(STARTSV) -SR(STARTSVS)*
     +.             MR(STARTR) + SI(STARTSVS)*MI(STARTR)
      .   SI(STARTSV) = SI(STARTSV) - SR(STARTSVS)*
     +.             MI(STARTR) -SI(STARTSVS)*MR(STARTR)
      .   STARTSV = STARTSV + K
      END FOR
      END
```

Second Back Substitution for LDL* or GE with L* Stored Columnwise

and $\overline{S}$ Componentwise

-L*W = T- 

(This method is identical to the Second Back Substitution with L* stored Rowwise and $\overline{S}$ Componentwise, except in the calculation of subscripts for MR and MI, which does not change the operation counts.)

Second Back Substitution for LL* with L* Stored Rowwise and $\overline{S}$ Vectorwise

$- L^*W = T -$ 

(see comments after LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* rowwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L*
T	FP	2N	temporary array used with SUM
DOT	FP	1	temporary location
LAST	I	1	location of last element of SCM
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location of last element of current steering vector
STARTR	I	1	location of ROW+1st element of ROWth row of the SCM
STARTSVS	I	1	location of ROW+1st element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1

```
      BEGIN
      STARTSV = N
      LAST = N*(N+1)/2 [LAST = N*N]

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO K
      .   STARTR = LAST -1 [STARTR = LAST - N]

C     *   CALCULATE NTH ELEMENT

      .   SR(STARTSV) = SR(STARTSV)*RECIP(N)
      .   SI(STARTSV) = SI(STARTSV)*RECIP(N)

C     *   CALCULATE N-1 COMPONENT

      .   SR(STARTSV-1) = SR(STARTSV-1) -MR(STARTR)*
     +.                   SR(STARTSV) + MI(STARTR)*SI(STARTSV)
      .   SI(STARTSV-1) = SI(STARTSV-1) -MR(STARTR)*
     +.                   SI(STARTSV) -MI(STARTR)*SR(STARTSV)
      .   SR(STARTSV-1) = SR(STARTSV-1)*RECIP(N-1)
      .   SI(STARTSV-1) = SI(STARTSV-1)*RECIP(N-1)
      .   STARTSVS = STARTSV-1
      .   LENGTH = 2
      .   FOR ROW = N-2 TO 1 STEP -1
      .   .   STARTR = STARTR - LENGTH -1 [STARTR = STARTR -N -1]
      .   .   T(1,LENGTH) = MR(STARTR,LENGTH)*
     +.   .                 SR(STARTSVS,LENGTH)
      .   .   T(LENGTH + 1,LENGTH) = - MI(STARTR,LENGTH)*
     +.   .                          SI(STARTSVS,LENGTH)
      .   .   DOT = SUM(T(1,2*LENGTH))
      .   .   SR(STARTSVS-1) = SR(STARTSVS-1) - DOT
      .   .   T(1,LENGTH) = MR(STARTR,LENGTH)*SI(STARTSVS,LENGTH)
      .   .   T(LENGTH + 1,LENGTH) = MI(STARTR,LENGTH)*SR(STARTSVS,LENGTH)
      .   .   DOT = SUM(T(1,2*LENGTH))
      .   .   SI(STARTSVS-1) = SI(STARTSVS -1)-DOT
      .   .   SR(STARTSVS-1) = SR(STARTSVS-1)*RECIP(ROW)
      .   .   SI(STARTSVS-1) = SI(STARTSVS-1)*RECIP(ROW)
      .   .   STARTSVS = STARTSVS-1
      .   .   LENGTH = LENGTH -1
      .   END FOR
      .   STARTSV = STARTSV + N
      END FOR
      END
```

Second Back Substitution for LL* with L* Stored Rowwise and $\overline{S}$ Componentwise

$-L^*W = T-$ 

(see comments under LDL* Decomposition

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ $[N^2]$	real part of L^* rowwise
MI	FP	$N(N+1)/2$ $[N^2]$	imaginary part of L* rowwise
SR	FP	KN	real parts of K steering vectors componentwise
SI	FP	KN	imaginary parts of K steering vectors componentwise
RECIP	FP	N	reciprocals of diagonal elements of L*
ROW	I	1	current row of SCM
STARTSV	I	1	beginning location of ROWth components of the steering vectors
SUBROW	I	1	row of components from which multiple of ROWth row is subtracted
STARTR	I	1	location of (SUBROW, ROW)th element of the SCM
STARTSVS	I	1	beginning location of $SUBROW^{th}$ components of the steering vectors

```
      BEGIN
      STARTSV = K*(N-1)+1

C     SUBTRACT MULTIPLES OF EACH COMPONENT FROM PREVIOUS COMPONENTS

      FOR ROW = N TO 2 STEP -1
      .   STARTR = ROW
      .   STARTSVS = 1
      .   SR(STARTSV,K) = SR(STARTSV,K)*RECIP(ROW)
      .   SI(STARTSV,K) = SI(STARTSV,K)*RECIP(ROW)
      .   FOR SUBROW = 1 TO ROW-1
      .   .   SR(STARTSVS,K) = SR(STARTSVS,K)
     +.   .                  -MR(STARTR)*SR(STARTSV,K)
     +.   .                  +MI(STARTR)*SI(STARTSV,K)
      .   .   SI(STARTSVS,K) = SI(STARTSVS,K)
     +.   .                  -MR(STARTR)*SI(STARTSV,K)
     +.   .                  -MI(STARTR)*SR(STARTSV,K)
      .   .   STARTR = STARTR + N -SUBROW [STARTR = STARTR+N]
      .   .   STARTSVS = STARTSVS + K
      .   END FOR
      .   STARTSV = STARTSV - K
      END FOR
      SR(1,K) = SR(1,K)*RECIP(1)
      SI(1,K) = SI(1,K)*RECIP(1)
      END
      (NOTE THAT K SHOULD BE GREATER THAN 1)
```

Second Back Substitution for LL* with L* Stored Columnwise and $\overline{S}$ Vectorwise

$-L^*T = S -$ 

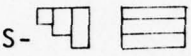
(see comments after LDL* Decomposition)

Variable	Type	Length	Contents
MR	FP	$N(N+1)/2$ [N^2]	real part of L* columnwise
MI	FP	$N(N+1)/2$ [N^2]	imaginary part of L* columnwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L*
LAST	I	1	location of the last element of SCM
AUG	I	1	current steering vector
STARTSV	I	1	location of first element of current steering vector
ROW	I	1	current column of SCM
STARTR	I	1	beginning location of current column of SCM
STARTSVS	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of current column of SCM excluding diagonal element

```
      BEGIN
C     ELIMINATE EACH STEERING VECTOR

      STARTSV = 1
      LAST = N*(N+1)/2 [LAST = N*N]
      FOR AUG = 1 TO K
      .   STARTSVS = STARTSV + K-1
      .   STARTR = LAST - N+1
      .   LENGTH = N-1
      .   FOR ROW = N TO 3 STEP -1
      .   .   SR(STARTSVS) = SR(STARTSVS)*RECIP(ROW)
      .   .   SI(STARTSVS) = SI(STARTSVS)*RECIP(ROW)
      .   .   SR(STARTSV,LENGTH) = SR(STARTSV,LENGTH)
     +.   .                -SR(STARTSVS)*MR(STARTR,LENGTH)
     +.   .                +SI(STARTSVS)*MI(STARTR,LENGTH)
      .   .   SI(STARTSV,LENGTH) = SI(STARTSV,LENGTH)
     +.   .                -SR(STARTSVS)*MI(STARTR,LENGTH)
     +.   .                -SI(STARTSVS)*MR(STARTR,LENGTH)
      .   .   STARTSVS = STARTSVS -1
      .   .   STARTR = STARTR -LENGTH
      .   .   LENGTH = LENGTH -1
      .   END FOR
C     *     CALCULATE 2ND ELEMENT OF STEERING VECTOR

      .   SR(STARTSVS) = SR(STARTSVS)*RECIP(2)
      .   SI(STARTSVS) = SI(STARTSVS)*RECIP(2)

C     *     CALCULATE 1ST ELEMENT OF STEERING VECTOR

      .   SR(STARTSV) = SR(STARTSV) -SR(STARTSVS)*
     +.             MR(STARTR) + SI(STARTSVS)*MI(STARTR)
      .   SI(STARTSV) = SI(STARTSV) -SR(STARTSVS)*
     +.             MI(STARTR) -SI(STARTSVS)*MR(STARTR)
      .   SR(STARTSV) = SR(STARTSV)*RECIP(1)
      .   SI(STARTSV) = SI(STARTSV)*RECIP(1)
      .   STARTSV = STARTSV+K
      END FOR
      END
```

Second Back Substitution for LL* with L* Stored Columnwise

and $\overline{S}$ Componentwise

$-L^*T = S-$ 

(This method is identical to the Second Back Substitution with L* stored Rowwise and $\overline{S}$ Componentwise, except in the calculation of subscripts of MR and MI, which does not affect the operation counts.)

Appendix C. CDC STAR-100 Software

SUMMARY AND DOCUMENTATION

The subroutine CHOLDC factors the positive definite Hermitian matrix M into its LL^* decomposition using the Cholesky algorithm. The factor L is stored in place of the input matrix M. Subroutine BAKSUB takes this lower triangular matrix L and a vector S as input and solves the system $MW=\overline{S}$ for W by performing two back substitutions.

CHOLDC

Algorithm: Since M is positive definite Hermitian it has a factorization of the form LL^* where L is a lower triangular matrix with real positive diagonal entries and L^* is its conjugate transpose, as discussed in the section on mathematical techniques. If N is the dimension of M, $M = \{m_{ij}\}$ and $L = \{\ell_{ij}\}$ we have

$$\ell_{ii} = \left[m_{ii} - \sum_{k=1}^{i-1} |\ell_{ik}|^2 \right]^{1/2} \qquad (1)$$

$$\ell_{ij} = \left[m_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\, \ell_{jk}^* \right] / \ell_{jj} \quad 1 \le j < i \qquad (2)$$

where the null sum $\sum_{k=1}^{0}$ is taken to be 0 and the square root is positive. Using these equations, one can successively solve for ℓ_{11}, the first column of L, ℓ_{22}, the second column of L,...,ℓ_{kk}, the k^{th} column of L,..., and ℓ_{NN}.

INPUTS:

NAME	TYPE	COMMON	CONTENTS
MATDIM	I	blank	Dimension of the matrix ≤ 200
REEL(20100)	R	blank	Real part of the lower half of matrix M stored columnwise: $m^R_{11}, m^R_{21}, \ldots m^R_{N1}, m^R_{22}, m^R_{32}, \ldots, m^R_{N2}, \ldots, m^R_{kk}, m^R_{k+1k}, \ldots, m^R_{NK}, \ldots m^R_{NN}$. where $m_{ij} = m^R_{ij} + \lambda m^R_{ij}$
JMAG(20100)	R	blank	Imaginary part of the lower half of the matrix M stored the same way as REEL.

OUTPUTS:

NAME	TYPE	COMMON	CONTENTS
REEL(20100)	I	blank	Real part of factor matrix L stored columnwise as above.
JMAG(20100)	I	blank	Imaginary part of factor matrix L stored as above.
RECIP(200)	I	blank	Reciprocals of diagonal elements of L (all positive real numbers by Equation (1)).

Comments (letters refer to labels in listing):

A. This routine was designed to work in a system with the following characteristics.

1) $MW=\bar{S}$ to be solved for arbitrarily many different S vectors for a given M matrix.
2) Frequent changes of matrix M (speed a paramount factor).
3) Infrequent changes of dimension.

The vague words "arbitrarily many", "frequent" and "infrequent" are used since they are a function of each system and may vary considerably. Our goal, however, is to find upper bounds in speed.

Since the changes in dimension are infrequent, we chose to compute the most frequently used vector descriptors once for each new dimension. MATOLD contains the old dimension and if it disagrees with the new one in MATDIM, new descriptors are calculated. Not all descriptors are calculated: temporary ones are assigned in lines 91 and 93 (numbers on left-hand side of page) of CHOLDC, since each one is only referenced once in a call to CHOLDC and there are almost as many of them as elements in the matrix ((MATDIM-2)* (MATDIM-1) versus MATDIM*(MATDIM+1), so storing them all would require a great deal of storage.

It is possible to attain a slight increase in speed in this section of code (at the expense of readability) by making the calculation of MATOLD, MTDMM1, MTDMM2, MTDMP1, and MTDMP2 a single vector addition and equivalencing the individual variable names to different elements of the vector. This may seem like a small point, but it shows how concerned we are about doing everything possible to speed up the program.

The following descriptor arrays are calculated for J=1,2,...,MATDIM-1:

NAME	CONTENTS
DREEL(J)	Points to j^{th} column of real part of matrix M (and later L) stored in array REEL.
DJMAG(J)	Points to j^{th} column of imaginary part of M and L in array JMAG.
DREEL1(J)	Points to j^{th} column excluding diagonal elements of real part of matrices M and L.
DJMAG1(J)	Points to j^{th} column excluding diagonal element of imaginary part of M and L.

NAME	CONTENTS
DTEMPR(J)	Points to the real part of the W vector in array TEMPR from the J+1st element to the end.
DTEMPJ(J)	Points to the imaginary part of the W vector in array TEMPJ from the J+1st element to the end.
DDTTMP(J)	Points to the first MATDIM-J elements of the temporary array DOTTMP.

The MATDIM-1st elements of these last four arrays are never used but are calculated to avoid an IF statement in the loop or another DO loop, under the assumption these alternatives would take longer.

These last four arrays are used primarily in subroutine BAKSUB but since BAKSUB may be called more than once (for different values of the S vector) for one call to CHOLDC, the calculation was done here to avoid unnecessary checking for changing dimensions, so that the instruction stack is not changing unnecessarily often. In the final implementation this function of calculating descriptors would probably be done by some executive routine.

B. Since only the reciprocals of diagonal elements of the L matrix are needed to both calculate the columns of the L matrix and perform the back substitutions, they are stored in array RECIP.

We assumed in coding that the compiler would optimize constant subscripts to be equivalent in speed to using equivalenced nonsubscripted variables. If this is not so, we would equivalence nonsubscripted variables to REEL(1), RECIP(1), DREEL(1), DJMAG(1) and to similar references elsewhere and use them instead.

Since we know the diagonal elements of L are real and only use their reciprocals anyway, it is unnecessary to calculate them and so we could just use DREEL1(1) and DJMAG1(1) here, shortening two vector multiplies by one element each.

C. The vector references DREEL(JCOL) and DJMAG(JCOL) are used in the loop described in section D, so we stored them in nonsubscripted variables DTR and DTJ in the hopes of making their use more efficient in the loop. This may, however, be unnecessary since the STAR has 255 registers, which, if used efficiently, could hold these descriptors for the duration of the loop. In other words, the compiler, upon first coming across the descriptor, would set aside a register for it for the duration of the loop, making the allocation of an extra memory location pointless. Similarly, the variables IFIRST, LEN, JCOLM1, MATDIM and the ones depending on MATDIM should be kept in registers.

D. The four values restored in scalars in the beginning of the loop were restored for the same reason as DREEL(JCOL) and DJAG(JCOL) in part C and the same comments apply here.

We know the imaginary part of the diagonal elements of L are 0, yet we calculate them here to avoid extra vector references. The alternative is:

```
Part C:     JFIRST=1
            LENM1=MTDMM1
            JCOLM1=0
            DO 3 JCOL=2,MTDMM1
               IFIRST=IFIRST+MTDMP2-JCOL
               ASSIGN DTR, DREEL(JCOL)
               ASSIGN DTJ, DJMAG1(JCOL)
               LEN=LENM1
               LENM1=LENM1-1
               ISUB=JCOL
               JCOLM1=JCOLM1+1

Part D:     DO 4 SUBCOL=1,JCOLM1
               TR=REEL(ISUB)
               TJ=JMAG(ISUB)
               ISUBP1=ISUB+1
               DTR=DTR+REEL(ISUB;LEN)*(-TR)
                       +JMAG(ISUB;LEN)*(-TJ)
               DTJ=DTJ+REEL(ISUBP1;LENM1)*(TJ)
                       +JMAG(ISUBP1;LENM1)+(-TR)
               ISUB=ISUB+MATDIM-SUBCOL
          4 CONTINUE
```

Calling this Method 2 to contrast it to Method 1 in the listing, we have the following table of "extra operations performed":

	Method 1	Method 2
Part C		1 scalar assignment
Part D	Increase in length by 1 of 2 vector multiplies and 2 vector adds	1 scalar addition 2 vector references

An intimate knowledge of the compiler is necessary to choose between methods.

Another possible change in this part is the use of explicit temporaries in the two vector assignment statements instead of letting the compiler allocate temporary space for the intermediate results of vector multiplications and additions. If a great deal of overhead is involved to allocate and free up space each time a temporary is needed, it might be more efficient to have some extra arrays like DOTTMP and do the allocation "manually."

E. As in part B we need not calculate the diagonal element of L, only its reciprocal, making it possible to shorten the length of the two vector assignment statements at the expense of having one and possibly two extra array references (DREEL1(JCOL) and possibly DJMAG1(JCOL)) instead of using the ones already available in DTR and DTJ.

F. There are two ways to perform the sum here, using the loop and subscript calculation as shown, or using double descriptors and the library function Q8SSUM, which has the added advantage of performing the sum in

AD-A054 358 TECHNOLOGY SERVICE CORP SANTA MONICA CALIF F/G 17/9
MULTIDOMAIN ALGORITHM EVALUATION. VOLUME II.(U)
APR 78 W C LILES, J C DEMMEL, I S REED F30602-76-C-0319
UNCLASSIFIED TSC-PD-B525-1-VOL-2 RADC-TR-78-59-VOL-2 NL

2 OF 3
AD A054358

double precision. The double descriptors would require a bit vector of length MATDIM*(MATDIM+1)/2 bits, and two vector multiplies (to calculate the sums of the squares of the real and imaginary parts) using it.

There are many different ways of summing the resulting squares. The two methods which come to mind immediately are

1) adding the vectors of squares and using Q8SSUM on the sum vector, and
2) using Q8SSUM twice on the vectors separately and adding the two scalar results.

Assuming that we do not need the double-precision results of Q8SSUM, we need only consider timing to choose algorithms.

Since the timing curve for vector additions as a function of vector length is a straight line not passing through the origin (because of the start up time) and the curve for Q8SSUM is probably similar, it may be that no one method may be best for all vector lengths. Some variant of the following method may be best (where the N elements of vector X are to be summed):

1) partition X into j consecutive subvectors, each of length $n_k, k=1,\ldots,j$; where $n_1 = \max_{1\le k\le j} n_k$

   $N = \sum_{k=1}^{j} n_k$

2) using vector addition, add the j-1 subvectors to the first one, resulting in a vector of length $n_1<N$, the sum of whose elements is the same as the sum of the elements of the original vector.

3) repeat steps 1) and 2) on the resulting shorter vector until one is obtained that is of the best length for summing by Q8SSUM.

There are obviously a lot of variables in the above algorithm. One specific example is to take the second half of the vector X and add it to the first half, repeating the bisection process and finally using Q8SSUM on the result. It requires an excellent knowledge of the timing of both Q8SSUM and vector addition as a function of vector length to optimize this algorithm.

BAKSUB:

Algorithm: Given the LL^* factorization of the matrix M, we can solve the system $MW=L(L^*W)=\bar{S}$ for W by solving the two triangular systems

$$L\bar{t} = S$$
$$L^*W = T$$

by back substitutions

$$t_i = \left[s_i - \sum_{j=1}^{i-1} \ell_{ij}\, t_j\right]/\ell_{ii} \qquad (3)$$

$$w_i = \left[t_i - \sum_{j=i+1}^{N} \ell^*_{ji}\, w_j\right]/\ell_{ii} \qquad (4)$$

where the null sums $\sum_{J=1}^{0}$ and $\sum_{J=N+1}^{N}$ are taken to be 0.

We successively compute $t_1, t_2, \ldots, t_{N-1}, t_N, w_N, w_{N+1}, \ldots, w_2, w_1$.

Inputs: In addition to the output from CHOLDC, BAKSUB uses:

NAME	TYPE	COMMON	CONTENTS
EREAL(200)	R	blank	Real part of vector S
EMAG(200)	R	blank	Imaginary part of vector S

Outputs:

NAME	TYPE	COMMON	CONTENTS
TEMPR(200)	R	work	Real part of vector W
TEMPJ(200)	R	work	Imaginary part vector W

COMMENTS

A. As in CHOLDC, the efficiency of this code depends on the compiler's treatment of constant array subscripts and vector temporaries.

B. Here we question whether the compiler saves XMUL in a register to use for both multiplies and whether it knows only to do one addition to address the array elements on either side of the equal sign.

C. Again we are concerned about the compiler's use of registers, vector temporaries, and identical array references on either side of the equal sign.

D. There are two alternate ways of calculating XMUL:

```
XMUL = RECIP(MATDIM)*RECIP(MATDIM)
```

and

```
XMUL = RECIP(MATDIM)
XMUL = XMUL*XMUL .
```

And to choose among the three methods requires a knowledge of how the compiler handles integer exponentiation and identical array references in one expression.

Also, if TEMPR(MTDMM1) and TEMPJ(MTDMM1) were kept in registers, there would be no need to store them in extra nonsubscripted variables TR1 and TJ1 for later use in lines 66, 67, 80, and 81.

Variables RS and JS are in COMMON/WORK/ and set equal to the next-to-last entries in REEL and JMAG, respectively, since these are calculated in CHOLDC and not changed in BAKSUB.

E. The first backsubstitution above used vector arithmetic since the coefficient matrix L was stored columnwise. In the second back substitution, the coefficient matrix, L^*, is stored row-wise, so we must use dot-products. In other words, in Equation (3), once t_j is calculated, $\ell_{ij}t_j$ is subtracted from s_i for i=j+1 through N in one vector operation; whereas, in Equation (4), the indicated dot-product of $\left\{\ell^*_{ji}\right\}_{j=i+1}^{N}$ and $\left\{w_j\right\}_{j=i+1}^{N}$ is actually performed. The efficiency of this code depends, as before, on

1) the speed of precalculated descriptors versus vector references,

2) implementation of vector temporaries such as DXR1, DXJ1, DTR, DTJ, and DDOT, and

3) speed of Q8SSUM,

all of which have been discussed before.

Finally, although this was not done in either routine, it must be remembered that vector addition is faster than just vector assignment, so that to copy vector B into vector A, it is faster to say

$$A = B + 0$$

than

$$A = B \ .$$

Figure C-1 Timing and Accuracy Results for Case MATDIM = 200

Dimension of matrix = 200
Time for descriptor calculation = 2.317 msec
Time for decomposition = 1400.990 msec

Error analysis comparing L (actual factor matrix) to $\tilde{L}$ (calculated factor matrix) (all errors absolute)

	Maximum Error	Location of Maximum Error	Average Error
Real part	.213E-6	20061	.734E-9
Imaginary part	.252E-6	20062	.694E-9
Absolute value	.294E-6	20062	.112E-8

Time for first backsubstitution = 18.273 msec
Time for second backsubstitution = 27.974 msec

Error analysis comparing W (actual solution) to $\tilde{W}$ (calculated solution) (all errors absolute)

	Maximum Error	Location of Maximum Error	Average Error
Real part	.512E-5	11	.278E-6
Imaginary part	.287E-5	19	.283E-6
Absolute value	.574E-5	11	.430E-6

```
      SUBROUTINE COVCMP
C     UPDATE SAMPLE COVARIANCE MATRIX
      COMMON /WCTIME/ WSETUP,WDECMP,WBKSB1,WBKSB2,WCVCMP
      COMMON /COV/ COVR(20100),COVJ(20100)
      COMMON /TIME/ TSETUP,TDECMP,TBKSB1,TBKSB2,TCVCMP
      COMMON /TEMP/ TDOT(200),SMPR(200),SMPJ(200),DTDOT,SRDOT,SIDOT,
     *WRDOT,WIDOT,WNRMR,WNRMI,SNRO
      DESCRIPTOR DTDOT,SRDOT,SIDOT,WRDOT,WIDOT,WNRMR,WNRMI
      COMMON MATDIM,REEL(20100),JMAG(20100),EREAL(200),EMAG(200)
      REAL JMAG
      DESCRIPTOR DRR1,DREEL,DJMAG,DSMPR,DSMPJ
      DIMENSION RR1(200)
      INTEGER W1,W2
      CALL Q8CLOCK(,,W1)
      T1=SECOND(X)
      J=1
      K=MATDIM
      DO 100 I=1,MATDIM
        ASSIGN DRR1,RR1(1;K)
        ASSIGN DREEL,COVR(J;K)
        ASSIGN DJMAG,COVJ(J;K)
        ASSIGN DSMPR,SMPR(I;K)
        ASSIGN DSMPJ,SMPJ(I;K)
        SR=SMPR(I)
        SJ=SMPJ(I)
        DRR1=DSMPR*SR
        DREEL=DREEL+DRR1
        DRR1=DSMPJ*SJ
        DREEL=DREEL+DRR1
        DRR1=DSMPJ*SR
        DJMAG=DJMAG-DRR1
        DRR1=DSMPR*SJ
        DJMAG=DJMAG+DRR1
        J=J+K
        K=K-1
  100 CONTINUE
      T2=SECOND(X)
      CALL Q8CLOCK(,,W2)
      WCVCMP=(W2-W1)/1000.
      TCVCMP=(T2-T1)*1000.
      RETURN
      END
```

```
43              SUBROUTINE CHOLDC

44        C
45        C     PERFORM CHOLESKY LL* DECOMPOSITION OF A POSITIVE DEFINITE
46        C     HERMITIAN MATRIX
47        C

48              COMMON /BLOWUP/ NEG
49              COMMON MATDIM,REEL(201ØØ),JMAG(201ØØ),EREAL(2ØØ),EMAG(2ØØ)
5Ø              REAL JMAG
51              COMMON /WORK/ MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
               *  DEREAL,DEMAG,RECIP(2ØØ),DREEL(2ØØ),DREEL1(2ØØ),DJMAG(2ØØ),
               *  DJMAG1(2ØØ),TEMPR(2ØØ),TEMPJ(2ØØ),DTEMPR(2ØØ),DTEMPJ(2ØØ),
               *  DOTTMP(2ØØ),DDTTMP(2ØØ),RS,JS
55              REAL JS
56              INTEGER SIZEM1
57              DESCRIPTOR DREEL1,DJMAG1,DTEMPR,DTEMPJ,DDTTMP,DEREAL,DEMAG
58              INTEGER SUBCOL,ESUB
59              DESCRIPTOR DREEL,DJMAG,DXR,DXJ,DTR,DTJ
6Ø              DESCRIPTOR DTTR,DTTJ
61              COMMON /TIME/ TSETUP,TDECMP,TBKSB1,TBKSB2,TCVCMP
62              COMMON /WCTIME/ WSETUP,WDECMP,WBKSB1,WBKSB2,WCVCMP
63              CALL Q8CLOCK( , ,IW1)
64              T1=SECOND(X)

65        C
66        C     SET UP ARRAY DESCRIPTORS
67        C

68              IF (MATDIM.EQ.MATOLD) GOTO 1
69                MATOLD=MATDIM
7Ø                MTDMM1=MATDIM-1
71                MTDMM2=MATDIM-2
72                MTDMP1=MATDIM+1
73                MTDMP2=MATDIM+2
74                ASSIGN DEREAL,EREAL(2;MTDMM1)
75                ASSIGN DEMAG,EMAG(2;MTDMM1)
76                ISUB=1
77                LENM1=MATDIM
78                LEN=MATDIM
79                DO 2 J=1,MTDMM1
8Ø                  LENM1=LENM1-1
81                  ISUBP1=ISUB+1
82                  JP1=J+1
83                  ASSIGN DREEL(J),REEL(ISUB;LEN)
84                  ASSIGN DJMAG(J),JMAG(ISUB;LEN)
85                  ASSIGN DDTTMP(J),DOTTMP(1;LENM1)
86                  ASSIGN DTEMPR(J),TEMPR(JP1;LENM1)
87                  ASSIGN DTEMPJ(J),TEMPJ(JP1;LENM1)
88                  ASSIGN DREEL1(J),REEL(ISUBP1;LENM1)
89                  ASSIGN DJMAG1(J),JMAG(ISUBP1;LENM1)
9Ø                  ISUB=ISUB+MTDMP1-J
91                  LEN=LENM1
92        2         CONTINUE
93                  SIZEM1=ISUB-1
94                  T2=SECOND(X)
```

```
 95              CALL Q8CLOCK( , ,IW2)
 96              TSETUP=( T2-T1)*1ØØØ.
 97              WSETUP=( IW2-IW1)/1ØØØ.
 98              RETURN
 99        1     CONTINUE
ØØ        C
Ø1        C     CALCULATE RECIPROCAL OF FIRST DIAGONAL ELEMENT
Ø2        C
Ø3              NEG=Ø
Ø4              IF ( REEL( 1).GT.Ø.Ø) GOTO 1Ø
Ø5                NONE=1
Ø6                WRITE ( 6,5ØØ) NONE,REEL( 1)
Ø7        5ØØ     FORMAT( 16HØERROR IN CHOLDC,I4,18H DIAGONAL ELEMENT=,E12.5)
Ø8                NEG=1
Ø9                RETURN
1Ø        1Ø    CONTINUE
11              XTEMP=1./SQRT( REEL( 1))
12              RECIP( 1)=XTEMP
13        C
14        C     CALCULATE FIRST COLUMN
15        C
16              DREEL( 1)=DREEL( 1)*XTEMP
17              DJMAG( 1)=DJMAG( 1)*XTEMP
18        C
19        C     CALCULATE COLUMNS 2 THROUGH MATDIM-1
2Ø        C
21              IFIRST=1
22              LEN=MATDIM
23              JCOLM1=Ø
24              DO 3 JCOL=2,MTDMM1
25                IFIRST=IFIRST+MTDMP2-JCOL
26                ASSIGN DTR,DREEL( JCOL)
27                ASSIGN DTJ,DJMAG( JCOL)
28                LEN=LEN-1
29                ISUB=JCOL
3Ø                JCOLM1=JCOLM1+1
31        C
32        C       SUBTRACT MULTIPLES OF PREVIOUS COLUMNS
33        C
34                DO 4 SUBCOL=1,JCOLM1
35                  TR=REEL( ISUB)
36                  ASSIGN DTTR,REEL( ISUB;LEN)
37                  TJ=JMAG( ISUB)
38                  ASSIGN DTTJ,JMAG( ISUB;LEN)
39                  DTR=DTR+DTTR*( -TR)+DTTJ*( -TJ)
4Ø                  DTJ=DTJ+DTTR*( TJ)+DTTJ*( -TR)
41                  ISUB=ISUB+MATDIM-SUBCOL
42        4       CONTINUE
```

B

C

D

```
C
C       CALCULATE RECIPROCAL OF DIAGONAL ELEMENT
C
          IF (REEL(IFIRST).GT.Ø.Ø) GOTO 11
            WRITE (6,5ØØ) JCOL,REEL(IFIRST)
            NEG=1
            RETURN
11        CONTINUE
          XTEMP=1./SQRT(REEL(IFIRST))
          RECIP(JCOL)=XTEMP
C
C       CALCULATE COLUMN
C
            DTR=DTR*XTEMP
            DTJ=DTJ*XTEMP
3         CONTINUE
C
C       CALCULATE LAST DIAGONAL ELEMENT
C
          SUM=Ø.
          ISUB=MATDIM
          DO 7 SUBCOL=1,MTDMM1
            XT=REEL(ISUB)
            XI=JMAG(ISUB)
            SUM=SUM+XT*XT+XI*XI
            ISUB=ISUB+MATDIM-SUBCOL
7         CONTINUE
          ASUM=REEL(ISUB)-SUM
          IF (ASUM.GT.Ø.Ø) GOTO 12
            WRITE (6,5ØØ) MATDIM,ASUM
            NEG=1
            RETURN
12        CONTINUE
          RECIP(MATDIM)=1./SQRT(ASUM)
          RS=REEL(SIZEM1)
          JS=JMAG(SIZEM1)
          T2=SECOND(X)
          CALL Q8CLOCK(,,IW2)
          TDECMP=(T2-T1)*1ØØØ.
          WDECMP=(IW2-IW1)/1ØØØ.
          RETURN
          END
```

E

F

```
185             SUBROUTINE BAKSUB
186       C
187       C     GIVEN THE LL* DECOMPOSITION OF A MATRIX, AND A VECTOR S, PERFORM
188       C     TWO BACK SUBSTITUTIONS TO SOLVE (LL*)W=S FOR W
189       C
190             COMMON /WCTIME/ WSETUP,WDECMP,WBKSB1,WBKSB2,WCVCMP
191             COMMON /TIME/ TSETUP,TDECMP,TBKSB1,TBKSB2,TCVCMP
192             COMMON MATDIM,REEL(20100),JMAG(20100),EREAL(200),EMAG(200)
193             COMMON /WORK/ MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
                *  DEREAL,DEMAG,RECIP(200),DREEL(200),DREEL1(200),DJMAG(200),
                *  DJMAG1(200),TEMPR(200),TEMPJ(200),DTEMPR(200),DTEMPJ(200),
                *  DOTTMP(200),DDTTMP(200),RS,JS
197             REAL JS
198             INTEGER SIZEM1
199             DESCRIPTOR DREEL,DJMAG
200             REAL JMAG
201             DESCRIPTOR DREEL1,DJMAG1,DTEMPR,DTEMPJ,DDTTMP,DEREAL,DEMAG
202             DESCRIPTOR DXR1,DXJ1,DTR,DTJ,DDOT
203             CALL Q8CLOCK(,,IW1)
204             T1=SECOND(X)
205       C
206       C     PERFORM FIRST BACK SUBSTITUTION, (L)TEMP=S
207       C
208       C     CALCULATE FIRST ELEMENT OF TEMP
209       C
210             XMUL=RECIP(1)
211             TEMPR(1)=EREAL(1)*XMUL
212             TEMPJ(1)=EMAG(1)*XMUL
213       C                                                              (A)
214       C     SUBTRACT MULTIPLE OF FIRST COLUMN OF L FROM S AND STORE IN TEMP
215       C
216             DTEMPR(1)=DEREAL+(-TEMPR(1))*DREEL1(1)+(TEMPJ(1))*DJMAG1(1)
217             DTEMPJ(1)=DEMAG+(-TEMPJ(1))*DREEL1(1)+(-TEMPR(1))*DJMAG1(1)
218       C
219       C     CALCULATE ELEMENTS 2 THROUGH MATDIM-2
220       C
221             DO 1 JCOL=2,MTDMM2
222       C
223       C       CALCULATE JCOL-TH ELEMENT
224       C
225               XMUL=RECIP(JCOL)
226               TEMPR(JCOL)=TEMPR(JCOL)*XMUL                         (B)
227               TEMPJ(JCOL)=TEMPJ(JCOL)*XMUL
228       C
229       C       SUBTRACT MULTIPLE OF JCOL-TH COLUMN FROM TEMP
```

```
230      C
231                ASSIGN DXR1,DREEL1(JCOL)
232                ASSIGN DXJ1,DJMAG1(JCOL)
233                TR=TEMPR(JCOL)
234                TJ=TEMPJ(JCOL)
235                DTEMPR(JCOL)=DTEMPR(JCOL)+(-TR)*DXR1+(TJ)*DXJ1
236                DTEMPJ(JCOL)=DTEMPJ(JCOL)+(-TJ)*DXR1+(-TR)*DXJ1
237      1       CONTINUE
238      C
239      C       CALCULATE ELEMENT MATDIM-1
240      C
241                XMUL=RECIP(MTDMM1)
242                TEMPR(MTDMM1)=TEMPR(MTDMM1)*XMUL
243                TEMPJ(MTDMM1)=TEMPJ(MTDMM1)*XMUL
244      C
245      C       COMBINE LAST STEP OF FIRST BACK SUBSTITUTION
246      C       (CALCULATION OF ELEMENT MATDIM) WITH FIRST STEP OF SECOND
247      C       BACK SUBSTITUTION (CALCULATION OF ELEMENT MATDIM OF W WHERE
248      C       (L*)*W=TEMP AND W IS STORED IN TEMP)
249      C
250                XMUL=RECIP(MATDIM)**2
251                TR1=TEMPR(MTDMM1)
252                TJ1=TEMPJ(MTDMM1)
253                TEMPR(MATDIM)=(TEMPR(MATDIM)-TR1*RS+TJ1*JS)*XMUL
254                TEMPJ(MATDIM)=(TEMPJ(MATDIM)-TJ1*RS-TR1*JS)*XMUL
255      C
256      C       PERFORM SECOND BACK SUBSTITUTION, (L*)W=TEMP, STORING W IN TEMP
257      C
258                T2=SECOND(X)
259                CALL Q8CLOCK(,,IW2)
260                T3=SECOND(X)
261      C
262      C       CALCULATE MATDIM-1-ST ELEMENT
263      C
264                XMUL=RECIP(MTDMM1)
265                TR=TEMPR(MATDIM)
266                TJ=TEMPJ(MATDIM)
267                TEMPR(MTDMM1)=(TR1-TR*RS-TJ*JS)*XMUL
268                TEMPJ(MTDMM1)=(TJ1-TJ*RS+TR*JS)*XMUL
269      C
270      C       CALCULATE ELEMENTS MATDIM-2 THROUGH 1
271      C
272                JROW=MTDMM1
273                DO 2 JTEMP=1,MTDMM2
274                  JROW=JROW-1
```

C

D

E

```
275       C
276       C        CALCULATE DOT PREDUCT OF JROW-TH ROW STARTING AT COLUMN JROW+1
277       C        WITH TEMP STARTING WITH THE JROW+1-ST ELEMENT
278       C

279                ASSIGN DXR1,DREEL1(JROW)
280                ASSIGN DXJ1,DJMAG1(JROW)
281                ASSIGN DTR,DTEMPR(JROW)
282                ASSIGN DTJ,DTEMPJ(JROW)
283                ASSIGN DDOT,DDTTMP(JROW)
284                DDOT=DXR1*DTR+DXJ1*DTJ
285                DOTR=Q8SSUM(DDOT)
286                DDOT=DXR1*DTJ-DXJ1*DTR
287                DOTJ=Q8SSUM(DDOT)

288       C
289       C        CALCULATE JROW-TH ELEMENT
290       C

291                XMUL=RECIP(JROW)
292                TEMPR(JROW)=(TEMPR(JROW)-DOTR)*XMUL
293                TEMPJ(JROW)=(TEMPJ(JROW)-DOTJ)*XMUL
294          2     CONTINUE
295                T4=SECOND(X)
296                CALL Q8CLOCK(,,IW3)
297                TBKSB1=(T2-T1)*1000.
298                TBKSB2=(T4-T3)*1000.
299                WBKSB1=(IW2-IW1)/1000.
300                WBKSB2=(IW3-IW2)/1000.
301                RETURN
302                END
```

```
303             PROGRAM SYSTST(TAPE5=INPUT,TAPE6=OUTPUT)

304       C     TEST COVARIANCE MATRIX APPROACH TO ADAPTIVE ARRAY PROBLEM

305             COMMON MATDIM,REEL(201ØØ),JMAG(201ØØ),EREAL(2ØØ),EMAG(2ØØ)
306             REAL JMAG
307             COMMON /BLOWUP/ NEG
308             COMMON /CPEXP/ NOLD,NOW,SAVEXP(2ØØ)
309             COMPLEX SAVEXP
310             COMMON /SETUP/ DIM,EIG(2ØØ),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(2ØØ)
311             REAL NOISE
312             INTEGER DIM
313             COMMON /DEBUG/ IDBG(2Ø)
314             REAL MXLGEG,MNLGEG
315             LOGICAL INOK
316             DATA MAXDIM/2ØØ/,MAXEIG/2ØØ/,MNLGEG/Ø./

317       C     INITIALIZE ROUTINE TO CALCULATE COMPLEX EXPONENTIALS

318             NOLD=Ø
319       999   CONTINUE

320       C     INPUT SYSTEM PARAMETERS
321       C     INPUT DIMENSION OF SYSTEM,NUMBER OF EIGENVALUES,COMMON LOG OF
322       C     MAXIMUM EIGENVALUE, LOOPING PARAMETERS FOR SAMPLE SIZE OF
323       C     SAMPLE COVARIANCE MATRIX,
324       C     DEBUG ARRAY--
325       C     (WHERE THE DEBUG FLAG INDICATES HOW MANY VALUES OF AN ARRAY
326       C      TO PRINT, -1 MEANS PRINT THE ENTIRE ARRAY, Ø MEANS PRINT NOTHING,
327       C      ^Ø MEANS PRINT THE INDICATED NUMBER OF ELEMENTS OR ROWS)
328       C     1  - NUMBER OF EIGENVALUES TO PRINT
329       C     2  - NUMBER OF ELEMENTS OF TRUE COVARIANCE MATRIX TO PRINT
330       C     3  - NUMBER OF OPTIMUM WEIGHTS TO PRINT
331       C     4  - NUMBER OF NORMALIZED OPTIMUM WEIGHTS TO PRINT
332       C     5  - CHECK IMAGINARY PART OF OPTIMUM SNR TO SEE IF ZERO
333       C     6  - NUMBER OF WEIGHTS FROM SAMPLES TO PRINT
334       C     7  - NUMBER OF NORMALIZED WEIGHTS FROM SAMPLES TO PRINT
335       C     8  - CHECK IMAGINARY PART OF SAMPLE SNR TO SEE IF ZERO
336       C     9  - NUMBER OF ROWS OF SAMPLE COVARIANCE MATRIX TO PRINT
337       C     1Ø - NUMBER OF DYNAMIC RANGES OF COMPONENTS OF SAMPLES TO PRINT
338       C     11 - NUMBER OF ROWS OF FACTORED TRUE MATRIX TO PRINT
339       C     12 - NUMBER OF ROWS OF FACTORED SAMPLE MATRIX TO PRINT
340       C     13 - NUMBER OF ELEMENTS OF EACH SAMPLE TO PRINT
341       C     14 - NUMBER OF COMPONENTS OF EACH VECTOR VOLTAGE TO PRINT
342       C     15 - IF .EQ. Ø STOP AFTER UNSUCCESSFUL DECOMPOSITION, ELSE GO ON

343             READ(5,1ØØ) DIM,NMEIG,MXLGEG,NSAMP1,NSAMP2,NSAMP3,NOISE(1),
                *  (IDBG(J),J=1,2Ø)
345       1ØØ   FORMAT(2I5,F1Ø.4,3I5,F16.1Ø/2ØI3)
346             IF (DIM.EQ.Ø) STOP 777
347             INOK=.TRUE.
348             IF ((DIM.GT.3).AND.(DIM.LE.MAXDIM)) GOTO 1
349               WRITE (6,1Ø1) DIM,MAXDIM
350       1Ø1     FORMAT(11HØDIMENSION=,I4,27H IS OUT OF RANGE 4 THROUGH ,I4)
351               INOK=.FALSE.
352       1     CONTINUE
```

```
353             IF ((NMEIG.GT.Ø).AND.(NMEIG.LE.MAXEIG)) GOTO 2
354               WRITE (6,1Ø2) NMEIG,MAXEIG
355       1Ø2     FORMAT(23HØNUMBER OF EIGENVALUES=,I4,16H IS OUT OF RANGE,
                *  11H 1 THROUGH ,I4)
357               INOK=.FALSE.
358       2       CONTINUE
359             IF (MXLGEG.GT.MNLGEG) GOTO 3
36Ø               WRITE (6,1Ø3) MXLGEG,MNLGEG
361       1Ø3     FORMAT(34HØCOMMON LOG OF MAXIMUM EIGENVALUE=,F1Ø.5,
                *  36H IS LESS THAN MINIMUM ALLOWED VALUE=,F1Ø.5)
363               INOK=.FALSE.
364       3       CONTINUE
365             IF (.NOT.INOK) STOP 1

366       C     CALCULATE EIGENVALUES

367             VAL=1Ø.**(MXLGEG/NMEIG)
368             EIGEN=1.
369             DO 4 J=1,NMEIG
37Ø               EIGEN=EIGEN*VAL
371               EIG(J)=EIGEN
372       4     CONTINUE

373       C     CALCULATE STEERING VECTOR

374             CALL STEER

375       C     CALCULATE RECEIVER NOISE

376             CALL RECNOS

377       C     DISPLAY INPUTS

378             WRITE(6,1Ø4)DIM,NMEIG,MXLGEG,NSAMP1,NSAMP2,NSAMP3,(IDBG(J),J=1,2Ø)
                *,(J,J=1,2Ø)
38Ø       1Ø4   FORMAT(14H1SYSTEM INPUTS,/,1ØHØDIMENSION,15(2H .),1X,I4,/,
                *22HØNUMBER OF EIGENVALUES,9(2H .),1X,I4,/,
                *33HØCOMMON LOG OF MAXIMUM EIGENVALUE,4(2H. ),F1Ø.5,/,
                *2ØHØINITIAL SAMPLE SIZE,1Ø(2H .),1X,I4,/,
                *18HØFINAL SAMPLE SIZE,11(2H .),1X,I4,/,
                *17HØSTEP SAMPLE SIZE,12(2H. ),I4,/,
                *12HØDEBUG ARRAY,14(2H .),1X,2ØI3,/,41X,2ØI3)
387             WRITE (6,1Ø5) (EREAL(J),J=1,DIM)
388       1Ø5   FORMAT(27HØSTEERING VECTOR(REAL PART),/,(1X,1ØE12.5))
389             WRITE (6,1Ø6) (EMAG(J),J=1,DIM)
39Ø       1Ø6   FORMAT(27HØSTEERING VECTOR(IMAG PART),/,(1X,1ØE12.5))
391             WRITE (6,1Ø7) (NOISE(J),J=1,DIM)
392       1Ø7   FORMAT(13HØNOISE VECTOR,/,(1X,1ØE12.5))
393             IF (IDBG(1).EQ.Ø) GOTO 5
394               LENP=IDBG(1)
395               IF (LENP.EQ.-1) LENP=NMEIG
396               WRITE (6,1Ø8) (EIG(J),J=1,LENP)
397       1Ø8     FORMAT(12HØEIGENVALUES,/,(1X,1ØE12.5))
398       5     CONTINUE

399       C     SOLVE PROBLEM WITH TRUE MATRIX, GET BEST SNR
```

```
400             CALL TSOLVE
401       C     IF DECOMPOSITION OF TRUE MATRIX FAILED, GET NEXT SET OF INPUTS
402             IF (NEG.EQ.1) GOTO 999
403       C     GENERATE SAMPLE MATRICES AND SOLVE FOR SNR
404             IF (NSAMP1.LE.NSAMP2) CALL SSOLVE
405       C     GO BACK FOR MORE DATA
406             GOTO 999
407             END
```

```
408             SUBROUTINE STEER
409       C     CALCULATE STEERING VECTOR
410             COMMON MATDIM,REEL(20100),JMAG(20100),EREAL(200),EMAG(200)
411             REAL JMAG
412             COMMON /SETUP/ DIM,EIG(200),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(200)
413             REAL NOISE
414             INTEGER DIM
415             EREAL(1;DIM)=1.
416             EMAG(1;DIM)=1.
417             RETURN
418             END
```

```
419             SUBROUTINE RECNOS

420       C     CALCULATE RECEIVER NOISE

421             COMMON /SETUP/ DIM,EIG(200),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(200)
422             REAL NOISE
423             INTEGER DIM

424       C *** CHANGED BY RITCHEY     10/10/77
425       C     DATA NOISE/200*1.E-6/

426             NOISE(2;DIM-1) = NOISE(1)
427             RETURN
428             END
```

```
429             SUBROUTINE TSOLVE

430       C     SOLVE SYSTEM USING TRUE COVARIANCE MATRIX FOR OPTIMUM SNR

431             COMMON /WCTIME/ WSETUP,WDECMP,WBKSB1,WBKSB2,WCVCMP
432             COMMON /BLOWUP/ NEG
433             COMMON /SETUP/ DIM,EIG(200),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(200)
434             REAL NOISE
435             INTEGER DIM
436             COMMON MATDIM,REEL(20100),JMAG(20100),EREAL(200),EMAG(200)
437             REAL JMAG
438             COMMON /WORK/ MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
               *  DEREAL,DEMAG,RECIP(200),DREEL(200),DREEL1(200),DJMAG(200),
               *  DJMAG1(200),TEMPR(200),TEMPJ(200),DTEMPR(200),DTEMPJ(200),
               *  DOTTMP(200),DDTTMP(200),RS,JS
442             REAL JS
443             INTEGER SIZEM1
444             DESCRIPTOR DREEL1,DJMAG1,DTEMPR,DTEMPJ,DDTTMP,DEREAL,DEMAG
445             DESCRIPTOR DREEL,DJMAG
446             COMMON /TIME/ TSETUP,TDECMP,TBKSB1,TBKSB2,TCVCMP
447             COMMON /DEBUG/ IDBG(20)
448             COMMON /TEMP/ TDOT(200),SMPR(200),SMPJ(200),DTDOT,SRDOT,SIDOT,
               *WRDOT,WIDOT,WNRMR,WNRMI,SNRO
450             DESCRIPTOR DTDOT,SRDOT,SIDOT,WRDOT,WIDOT,WNRMR,WNRMI
451             COMMON /SAVEW/ WRN(200),WIN(200)

452       C     CALCULATE TRUE COVARIANCE MATRIX

453             CALL BUIDTM

454       C     SOLVE SYSTEM
455       C     SET UP DESCRIPTORS

456             MATDIM=DIM
457             MATOLD=DIM+1
458             CALL CHOLDC
459             IF (IDBG(11).NE.0) WRITE (6,108)
460       108   FORMAT(27H0DESCRIPTOR SET UP COMPLETE)

461       C     PERFORM LL* DECOMPOSITION

462             CALL CHOLDC

463       C     IF DECOMPOSITION FAILED, PRINT OUT MATRIX AND RETURN FOR NEXT
464       C     SET OF SYSTEM INPUTS

465             IF (NEG.EQ.1) IDBG(11)=-1
466             IF (IDBG(11).EQ.0) GOTO 6
467               IDEBUG=IDBG(11)
468               IF (IDEBUG.EQ.-1) IDEBUG=DIM
469               WRITE (6,109)
470       109     FORMAT(32H0FACTORED TRUE MATRIX(REAL PART))
471               CALL DISP(DIM,REEL,IDEBUG)
472               WRITE (6,110)
473       110     FORMAT(32H0FACTORED TRUE MATRIX(IMAG PART))
474               CALL DISP(DIM,JMAG,IDEBUG)
```

```
475                 WRITE (6,112) (RECIP(K),K=1,IDEBUG)
476       112       FORMAT(26HØTRUE RECIPROCAL DIAGONALS,/,(1X,1ØE12.5))
477       6       CONTINUE
478               IF (NEG.EQ.Ø) GOTO 1Ø
479                 WRITE (6,111)
48Ø       111       FORMAT(1HØ,1Ø(1H*),35HDECOMPOSITION OF TRUE MATRIX FAILED)
481                 RETURN
482       1Ø      CONTINUE

483       C       CONJUGATE STEERING VECTOR

484               EMAG(1;DIM)=-EMAG(1;DIM)

485       C       PERFORM BACK SUBSTITUTIONS

486               CALL BAKSUB
487               IF (IDBG(3).EQ.Ø) GOTO 1
488                 LENP=IDBG(3)
489                 IF (LENP.EQ.-1) LENP=MATDIM
49Ø                 WRITE (6,1ØØ) (TEMPR(J),J=1,LENP)
491       1ØØ       FORMAT(27HØOPTIMUM WEIGHTS(REAL PART),/,(1X,1ØE12.5))
492                 WRITE (6,1Ø1) (TEMPJ(J),J=1,LENP)
493       1Ø1       FORMAT(27HØOPTIMUM WEIGHTS(IMAG PART),/,(1X,1ØE12.5))
494       1       CONTINUE

495       C       NORMALIZE WEIGHTS SO THEIR ONE-NORM IS ONE AND SAVE THEM

496               SUM=Ø
497               DO 2 J=1,DIM
498                 SUM=SUM+SQRT(TEMPR(J)**2+TEMPJ(J)**2)
499       2       CONTINUE
5ØØ               XMUL=1./SUM
5Ø1               WRN(1;DIM)=TEMPR(1;DIM)*XMUL
5Ø2               WIN(1;DIM)=TEMPJ(1;DIM)*XMUL
5Ø3               IF (IDBG(4).EQ.Ø) GOTO 4
5Ø4                 WRITE (6,1Ø2) SUM
5Ø5       1Ø2       FORMAT(29HØONE-NORM OF OPTIMUM WEIGHTS=,E12.5)
5Ø6                 LENP=IDBG(4)
5Ø7                 IF (LENP.EQ.-1) LENP=MATDIM
5Ø8                 WRITE (6,1Ø3) (WRN(J),J=1,LENP)
5Ø9       1Ø3       FORMAT(38HØNORMALIZED OPTIMUM WEIGHTS(REAL PART),/,(1X,1ØE12.5))
51Ø                 WRITE (6,1Ø4) (WIN(J),J=1,LENP)
511       1Ø4       FORMAT(38HØNORMALIZED OPTIMUM WEIGHTS(IMAG PART),/,(1X,1ØE12.5))
512       4       CONTINUE

513       C       CALCULATE MAXIMUM POSSIBLE TIMES

514               TIMMAX=TSETUP+TDECMP+TBKSB1+TBKSB2
515               WCTMAX=WSETUP+WDECMP+WBKSB1+WBKSB2

516       C       CALCUALTE OPTIMUM SNR

517               ASSIGN DTDOT,TDOT(1;DIM)
518               ASSIGN WNRMR,SMPR(1;DIM)
519               ASSIGN WNRMI,SMPJ(1;DIM)
52Ø               ASSIGN SRDOT,EREAL(1;DIM)
```

```
521             ASSIGN SIDOT,EMAG(1;DIM)
522             ASSIGN WRDOT,TEMPR(1;DIM)
523             ASSIGN WIDOT,TEMPJ(1;DIM)
524             DTDOT=SRDOT*WRDOT+SIDOT*WIDOT
525             SNRO=Q8SSUM(DTDOT)
526             WRITE (6,105)
527       105   FORMAT(56H0 NUMBER        SNR DB DOWN  TIME FOR  TIME FOR  TIME FOR
                1 ,60H  TIME FOR  TIME PER   TIME TO   MAXIMUM  ONE-NORM  SUP-NORM,
                2 10H  LOCATION,/,
                3 56H SAMPLES                        SET UP    DECOMP BACKSUB 1,
                4 60H BACKSUB 2 SAMPLE TO      MOVE  POSSIBLE  OF ERROR  OF ERROR,
                5 10H         OF,/,
                6 26X,4(10H  MILLISEC),30H    UPDATE    MATRIX      TIME,20X,
                7 10H  SUP-NORM,/,
                8 26X,4(10H     (CPU)),30H    MATRIX  MILLISEC  MILLISEC,/,
                9 26X,4(10H    (WALL)),30H  MILLISEC     (CPU)     (CPU),/,
                * 66X,30H     (CPU)    (WALL)    (WALL),/,
                1 66X,10H    (WALL))
539             WRITE (6,107) SNRO,TSETUP,TDECMP,TBKSB1,TBKSB2,TIMMAX,
                *  WSETUP,WDECMP,WBKSB1,WBKSB2,WCTMAX
541       107   FORMAT(8H0OPTIMUM,E10.3,8X,4F10.3,20X,F10.3,/,
                *  26X,4F10.3,20X,F10.3)
543             IF (IDBG(5).EQ.0) GOTO 5
544               DTDOT=SRDOT*WIDOT-SIDOT*WRDOT
545               TSNR=Q8SSUM(DTDOT)
546               WRITE (6,106) TSNR
547       106     FORMAT(26H0IMAG PART OF OPTIMUM SNR=,E12.5)
548       5     CONTINUE
549             RETURN
550             END
```

```
551             SUBROUTINE DOEXP

552       C     PRECALCULATE COMPLEX EXPONENTIALS
553       C     (NOW-TH ROOTS OF UNITY)

554             COMMON /CPEXP/ NOLD,NOW,SAVEXP(2ØØ)
555             COMPLEX SAVEXP
556             COMPLEX MUL,PROD
557             DATA PI/3.14159265358979323/
558             NOLD=NOW
559             MUL=CEXP(CMPLX(Ø.Ø,2.Ø*PI/FLOAT(NOW)))
56Ø             PROD=CMPLX(1.Ø,Ø.Ø)
561             DO 1 J=1,NOW
562               SAVEXP(J)=PROD
563               PROD=PROD*MUL
564       1     CONTINUE
565             RETURN
566             END
```

```
567             SUBROUTINE BUIDTM

568       C     FORM TRUE COVARIANCE MATRIX

569             COMMON /CPEXP/ NOLD,NOW,SAVEXP(200)
570             COMPLEX SAVEXP
571             COMMON /SETUP/ DIM,EIG(200),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(200)
572             REAL NOISE
573             INTEGER DIM
574             COMMON MATDIM,REEL(20100),JMAG(20100),EREAL(200),EMAG(200)
575             REAL JMAG
576             COMMON /DEBUG/ IDBG(20)
577             COMMON /TRUE/ ER(200),EC(200)
578             COMPLEX S
579             INTEGER DIMM1,DIMP1

580       C     PRECALCULATE COMPLEX EXPONENTIALS

581             NOW=DIM
582             IF (NOW.NE.NOLD) CALL DOEXP

583       C     COMPUTE EXP FACTORS OF LOWER HALF OF TOEPLETZ COVARIANCE MATRIX

584             SUM=0.0
585             DO 100 K=1,NMEIG
586               SUM=SUM+EIG(K)
587       100   CONTINUE
588             ER(1)=SUM
589             EC(1)=0.0
590             DIMM1=DIM-1
591             DO 200 I=1,DIMM1
592               ISUB=I+1
593               S=CMPLX(0.0,0.0)
594               DO 210 K=1,NMEIG
595                 S=S+EIG(K)*SAVEXP(ISUB)
596                 ISUB=ISUB+I
597                 IF (ISUB.GT.DIM) ISUB=ISUB-DIM
598       210     CONTINUE
599               ER(I+1)=REAL(S)
600               EC(I+1)=-AIMAG(S)
601       200   CONTINUE
602             IF (IDBG(2).EQ.0) GOTO 500
603               LENP=IDBG(2)
604               IF (LENP.EQ.-1) LENP=DIM
605               WRITE (6,808) (ER(I),I=1,LENP)
606       808     FORMAT(23H0TRUE MATRIX(REAL PART),/,(1X,10E12.5))
607               WRITE (6,809) (EC(I),I=1,LENP)
608       809     FORMAT(23H0TRUE MATRIX(IMAG PART),/,(1X,10E12.5))
609       500   CONTINUE

610       C     FORM COVARIANCE MATRIX

611             DIMP1=DIM+1
612             INX=0
613             DO 300 I=1,DIM
614               LOOP=DIMP1-I
```

```
615                 DO 310 J=1,LOOP
616                   INX=INX+1
617                   REEL(INX)=ER(J)
618                   JMAG(INX)=EC(J)
619         310     CONTINUE
620         300   CONTINUE

621         C     ADD NOISE TO DIAGONAL

622               J=1
623               DO 400 I=1,DIM
624                 REEL(J)=REEL(J)+NOISE(I)
625                 J=J+DIMP1-I
626         400   CONTINUE
627               RETURN
628               END
```

```
629             SUBROUTINE SSOLVE

630       C     SOLVE SYSTEM USING SAMPLE COVARIANCE MATRIX

631             COMMON /SMPHLP/ IDBG14,NUMSAM
632             COMMON /WCTIME/ WSETUP,WDECMP,WBKSB1,WBKSB2,WCVCMP
633             COMMON MATDIM,REEL(201ØØ),JMAG(201ØØ),EREAL(2ØØ),EMAG(2ØØ)
634             REAL JMAG
635             COMMON /BLOWUP/ NEG
636             COMMON /COV/ COVR(201ØØ),COVJ(201ØØ)
637             COMMON /SETUP/ DIM,EIG(2ØØ),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(2ØØ)
638             REAL NOISE
639             INTEGER DIM
640             COMMON /WORK/ MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
                *  DEREAL,DEMAG,RECIP(2ØØ),DREEL(2ØØ),DREEL1(2ØØ),DJMAG(2ØØ),
                *  DJMAG1(2ØØ),TEMPR(2ØØ),TEMPJ(2ØØ),DTEMPR(2ØØ),DTEMPJ(2ØØ),
                *  DOTTMP(2ØØ),DDTTMP(2ØØ),RS,JS
644             REAL JS
645             INTEGER SIZEM1
646             DESCRIPTOR DREEL1,DJMAG1,DTEMPR,DTEMPJ,DDTTMP,DEREAL,DEMAG
647             DESCRIPTOR DREEL,DJMAG
648             COMMON /TIME/ TSETUP,TDECMP,TBKSB1,TBKSB2,TCVCMP
649             COMMON /TRUE/ ER(2ØØ),EC(2ØØ)
650             COMMON /TEMP/ TDOT(2ØØ),SMPR(2ØØ),SMPJ(2ØØ),DTDOT,SRDOT,SIDOT,
               *WRDOT,WIDOT,WNRMR,WNRMI,SNRO
652             DESCRIPTOR DTDOT,SRDOT,SIDOT,WRDOT,WIDOT,WNRMR,WNRMI
653             COMMON /DEBUG/ IDBG(2Ø)
654             COMMON /SAVEW/ WRN(2ØØ),WIN(2ØØ)
655             DIMENSION SMIN(2ØØ),SMAX(2ØØ)
656             INTEGER WT1,WT2
657             COMPLEX CRANG,CRANGI,RAN
658             INTEGER SAMTOT
659             DESCRIPTOR DCOVR,DCOVJ
660             DESCRIPTOR DDR,DDJ
661             DATA PI/3.141592653589793 23/
662             DATA INTRND/Ø/

663       C     INITIALIZE RANDOM NUMBER GENERATOR

664             IF (INTRND.EQ.1) GOTO 5ØØ
665               INTRND=1
666               RAN=CRANGI(Ø.Ø)
667       5ØØ   CONTINUE

668       C     INITIALIZE SAMPLE GENERATING ROUTINE

669             CALL SAMPI

670       C     ZERO OUT COVARIANCE MATRIX

671             LEN=DIM*(DIM+1)/2
672             ASSIGN DDR,REEL(1;LEN)
673             ASSIGN DDJ,JMAG(1;LEN)
674             ASSIGN DCOVR,COVR(1;LEN)
675             ASSIGN DCOVJ,COVJ(1;LEN)
676             DCOVR=Ø.Ø
```

```
677             DCOVJ=Ø.Ø
678             IDBG14=IDBG(14)
679             IF (IDBG14.EQ.-1) IDBG14=NMEIG
68Ø             IDEBUG=IDBG(1Ø)
681             IF (IDEBUG.EQ.-1) IDEBUG=DIM
682             TUPDAT=Ø.Ø
683             WUPDAT=Ø.Ø
684             SAMTOT=Ø
685             KDEBUG=IDBG(13)
686             IF (KDEBUG.EQ.-1) KDEBUG=DIM

687       C     LOOP THROUGH SAMPLE SIZES

688             DO 1 NSAMP=NSAMP1,NSAMP2,NSAMP3
689               NEED=NSAMP-SAMTOT

69Ø       C       LOOP THROUGH NUMBER OF EXTRA SAMPLES NEEDED

691               TTCOV=Ø.
692               WTCOV=Ø.Ø
693               IF (IDEBUG.EQ.Ø) GOTO 13
694                 SMIN(1;IDEBUG)=1.E3Ø
695                 SMAX(1;IDEBUG)=-1.E3Ø
696       13        CONTINUE
697               LOPMIN=SAMTOT+1
698               DO 3 J=LOPMIN,NSAMP

699       C         GET SAMPLE

7ØØ                 NUMSAM=J
7Ø1                 CALL SAMP
7Ø2                 IF (IDEBUG.EQ.Ø) GOTO 122
7Ø3                   DO 123 K=1,IDEBUG
7Ø4                     SABS=SQRT(SMPR(K)**2+SMPJ(K)**2)
7Ø5                     SMIN(K)=AMIN1(SMIN(K),SABS)
7Ø6                     SMAX(K)=AMAX1(SMAX(K),SABS)
7Ø7       123           CONTINUE
7Ø8       122         CONTINUE
7Ø9                   IF (KDEBUG.EQ.Ø) GOTO 44
71Ø                     WRITE (6,441) J,(SMPR(JJ),JJ=1,KDEBUG)
711       441           FORMAT(14HØSAMPLE NUMBER,I4,12H (REAL PART),/,(1X,1ØE12.5))
712                     WRITE (6,442) J,(SMPJ(JJ),JJ=1,KDEBUG)
713       442           FORMAT(14HØSAMPLE NUMBER,I4,12H (IMAG PART),/,(1X,1ØE12.5))
714       44          CONTINUE

715       C           ADD TO COVARIANCE MATRIX

716                   CALL COVCMP
717                   TTCOV=TTCOV+TCVCMP
718                   WTCOV=WTCOV+WCVCMP
719       3         CONTINUE
72Ø                 IF (IDEBUG.EQ.Ø) GOTO 14
721                   WRITE (6,11Ø) NEED,(KS,SMIN(KS),SMAX(KS),KS=1,IDEBUG)
722       11Ø         FORMAT(32HØDYNAMIC RANGE OF COMPONENTS FOR,I4,8H SAMPLES,
                *     (/,1X,I3,1X,E1Ø.4,1X,E1Ø.4,1X,I3,1X,E1Ø.4,1X,E1Ø.4,1X,I3,1X,
                *     E1Ø.4,1X,E1Ø.4,1X,I3,1X,E1Ø.4,1X,E1Ø.4,1X,I3,1X,E1Ø.4,1X,
```

```
     *    E10.4))
726  14   CONTINUE

727  C    MOVE DATA FROM (COVR,COVJ) TO (REEL,JMAG)

728       CALL Q8CLOCK(,,WT1)
729       TT1=SECOND(X)
730       DDR=DCOVR
731       DDJ=DCOVJ
732       TT2=SECOND(X)
733       CALL Q8CLOCK(,,WT2)
734       TMOVE=1000.*(TT2-TT1)
735       WMOVE=(WT2-WT1)/1000.
736       SAMTOT=NSAMP
737       IF (IDBG(9).EQ.0) GOTO 12
738         WRITE (6,108) NSAMP
739  108    FORMAT(13H0SAMPLE SIZE=,I4,5X,24HSAMPLE COVARIANCE MATRIX,
     *    11H(REAL PART))
741         LENP=IDBG(9)
742         IF (LENP.EQ.-1) LENP=DIM
743         CALL DISP(MATDIM,REEL,LENP)
744         WRITE (6,109) NSAMP
745  109    FORMAT(13H0SAMPLE SIZE=,I4,5X,24HSAMPLE COVARIANCE MATRIX,
     *    11H(IMAG PART))
747         CALL DISP(MATDIM,JMAG,LENP)
748         WRITE (6,113) NSAMP,(RECIP(K),K=1,LENP)
749  113    FORMAT(11H0FOR NSAMP=,I5,21H RECIPROCAL DIAGONALS,/,(1X,10E12
     *    .5))
751  12     CONTINUE

752  C      SOLVE SYSTEM WITH SAMPLE COVARIANCE MATRIX
753  C      PERFORM LL* DECOMPOSITION

754       CALL CHOLDC

755  C      IF DECOMPOSITION FAILED, PRINT OUT MATRIX AND GET MORE SAMPLES

756       IF (NEG.EQ.0) GOTO 45
757         IDTMP=IDBG(12)
758         IDBG(12)=-1
759  45     CONTINUE
760       IF (IDBG(12).EQ.0) GOTO 20
761         IDEBUG=IDBG(12)
762         IF (IDEBUG.EQ.-1) IDEBUG=DIM
763         WRITE (6,111) NSAMP
764  111    FORMAT(45H0FACTORED SAMPLE MATRIX(REAL PART) FOR NSAMP=,I5)
765         CALL DISP(DIM,REEL,IDEBUG)
766         WRITE (6,112) NSAMP
767  112    FORMAT(45H0FACTORED SAMPLE MATRIX(IMAG PART) FOR NSAMP=,I5)
768         CALL DISP(DIM,JMAG,IDEBUG)
769  20     CONTINUE
770       IF (NEG.EQ.0) GOTO 46
771         IDBG(12)=IDTMP
772         WRITE (6,47) NSAMP
773  47     FORMAT(1H0,110,5(1H*),20HDECOMPOSITION FAILED,20(1H*))
774         IF (IDBG(15).EQ.0) RETURN
```

```
775                 GOTO 1
776       46        CONTINUE

777       C         PERFORM BACK SUBSTITUTIONS

778                 CALL BAKSUB
779                 IF (IDBG(6).EQ.Ø) GOTO 7
78Ø                   LENP=IDBG(6)
781                   IF (LENP.EQ.-1) LENP=DIM
782                   WRITE (6,1ØØ) NSAMP,(TEMPR(J),J=1,LENP)
783       1ØØ         FORMAT(36HØWEIGHTS(REAL PART) FOR SAMPLE SIZE=,I1Ø,
                *     /,(1X,1ØE12.5))
785                   WRITE (6,1Ø1) NSAMP,(TEMPJ(J),J=1,LENP)
786       1Ø1         FORMAT(36HØWEIGHTS(IMAG PART) FOR SAMPLE SIZE=,I1Ø,
                *     /,(1X,1ØE12.5))
788       7         CONTINUE

789       C         CALCULATE SNR, USE SMPR AND SMPJ FOR TEMPORARY STORAGE

79Ø                 DTDOT=WRDOT*SRDOT+WIDOT*SIDOT
791                 XNUM=(Q8SSUM(DTDOT))**2
792                 DTDOT=-WRDOT*SIDOT+WIDOT*SRDOT
793                 XNUM=XNUM+(Q8SSUM(DTDOT))**2
794                 DO 4Ø JSNR=1,DIM
795                   TSNRR=Ø.
796                   TSNRI=Ø.
797                   JSNRP1=JSNR+1
798                   JSNRM1=JSNR-1
799                   IF (JSNR.EQ.1) GOTO 41
8ØØ                     DO 42 KSNR=1,JSNRM1
8Ø1                       TSNRR=TSNRR+TEMPR(KSNR)*ER(JSNRP1-KSNR)-
                *         TEMPJ(KSNR)*EC(JSNRP1-KSNR)
8Ø3                       TSNRI=TSNRI+TEMPR(KSNR)*EC(JSNRP1-KSNR)+
                *         TEMPJ(KSNR)*ER(JSNRP1-KSNR)
8Ø5       42              CONTINUE
8Ø6       41        CONTINUE
8Ø7                 TSNRR=TSNRR+TEMPR(JSNR)*(ER(1)+NOISE(JSNR))
8Ø8                 TSNRI=TSNRI+TEMPJ(JSNR)*(ER(1)+NOISE(JSNR))
8Ø9                 IF (JSNR.EQ.DIM) GOTO 48
81Ø                   DO 43 KSNR=JSNRP1,DIM
811                     TSNRR=TSNRR+TEMPR(KSNR)*ER(KSNR-JSNRM1)+
                *       TEMPJ(KSNR)*EC(KSNR-JSNRM1)
813                     TSNRI=TSNRI-TEMPR(KSNR)*EC(KSNR-JSNRM1)+
                *       TEMPJ(KSNR)*ER(KSNR-JSNRM1)
815       43            CONTINUE
816       48        CONTINUE
817                 SMPR(JSNR)=TSNRR
818                 SMPJ(JSNR)=TSNRI
819       4Ø        CONTINUE
82Ø                 DTDOT=WRDOT*WNRMR+WIDOT*WNRMI
821                 XDENR=Q8SSUM(DTDOT)
822                 DTDOT=WRDOT*WNRMI-WIDOT*WNRMR
823                 XDENI=Q8SSUM(DTDOT)
824                 XDEN=XDENR*XDENR+XDENI*XDENI
825                 SNR=XNUM*XDENR/XDEN
826                 IF (IDBG(8).EQ.Ø) GOTO 1Ø
```

```
827                 SNRI=-XNUM*XDENI/XDEN
828                 WRITE (6,105) NSAMP,SNRI
829       105       FORMAT(17HØFOR SAMPLE SIZE=,I10,18H IMAG PART OF SNR=,E12.5)
830       10        CONTINUE

831       C         NORMALIZE WEIGHTS, STORE IN SMPR,SMPJ

832                 SUM=Ø.Ø
833                 DO 8 J=1,DIM
834                   SUM=SUM+SQRT(TEMPR(J)**2+TEMPJ(J)**2)
835       8         CONTINUE
836                 XMUL=1./SUM
837                 WNRMR=WRDOT*XMUL
838                 WNRMI=WIDOT*XMUL
839                 IF (IDBG(7).EQ.Ø) GOTO 9
840                   LENP=IDBG(7)
841                   IF (LENP.EQ.-1) LENP=DIM
842                   WRITE (6,102) NSAMP,SUM
843       102         FORMAT(13HØSAMPLE SIZE=,I10,5X,20HONE-NORM OF WEIGHTS=,E12.5)
844                   WRITE (6,103) (SMPR(J),J=1,LENP)
845       103         FORMAT(30HØNORMALIZED WEIGHTS(REAL PART),/,(1X,10E12.5))
846                   WRITE(6,104) (SMPJ(J),J=1,LENP)
847       104         FORMAT(30HØNORMALIZED WEIGHTS(IMAG PART),/,(1X,10E12.5))
848       9         CONTINUE

849       C         PERFORM OTHER ERROR ANALYSIS
850       C         CALCULATE ONE-NORM OF ERROR(ABSOLUTE-RELATIVE)

851                 EMAX=-1.E30
852                 LMAX=1
853                 SUM=Ø.Ø
854                 DO 11 J=1,DIM
855                   DIFR=WRN(J)-SMPR(J)
856                   DIFI=WIN(J)-SMPJ(J)
857                   ERQ=SQRT(DIFR*DIFR+DIFI*DIFI)
858                   SUM=SUM+ERQ
859                   IF (ERQ.LT.EMAX) GOTO 11
860                     EMAX=ERQ
861                     LMAX=J
862       11        CONTINUE
863                 TUPDAT=TUPDAT+TTCOV
864                 WUPDAT=WUPDAT+WTCOV
865                 TTCOV=TTCOV/NEED
866                 WTCOV=WTCOV/NEED
867                 TIMMAX=TUPDAT+TMOVE+TSETUP+TDECMP+TBKSB1+TBKSB2
868                 WCTMAX=WUPDAT+WMOVE+WSETUP+WDECMP+WBKSB1+WBKSB2
869                 DBDOWN=10.*ALOG10(SNRO/SNR)
870                 WRITE (6,107) NSAMP,SNR,DBDOWN,TDECMP,TBKSB1,TBKSB2,TTCOV,TMOVE,
                *   TIMMAX,SUM,EMAX,LMAX,WDECMP,WBKSB1,WBKSB2,WTCOV,WMOVE,WCTMAX
872       107       FORMAT(1H0,I7,E10.3,F8.3,10X,8F10.3,I10,/,36X,6F10.3)
873       1       CONTINUE
874               RETURN
875               END
```

```
876             SUBROUTINE SAMP
877       C     GENERATE A RANDOM VECTOR WHOSE COMPONENTS HAVE THE COVARIANCE
878       C     MATRIX DEFINED IN BUIDTM
879             COMMON /SMPHLP/ IDBG14,NUMSAM
880             COMMON /CPEXP/ NOLD,NOW,SAVEXP(200)
881             COMPLEX SAVEXP
882             COMMON /SETUP/ DIM,EIG(200),NMEIG,NSAMP1,NSAMP2,NSAMP3,NOISE(200)
883             REAL NOISE
884             INTEGER DIM
885             COMMON /DEBUG/ IDBG(20)
886             COMMON /TEMP/ TDOT(200),SMPR(200),SMPJ(200),DTDOT,SRDOT,SIDOT,
               *WRDOT,WIDOT,WNRMR,WNRMI,SNRO
888             DESCRIPTOR DTDOT,SRDOT,SIDOT,WRDOT,WIDOT,WNRMR,WNRMI
889             DIMENSION SQEIG(200),RD(200),SQNOS(200)
890             COMPLEX RD,SUM
891             COMPLEX CRANG
892       C     CALCULATE RANDOM VOLTAGES FOR EACH EIGENVALUE
893             DO 1 K=1,NMEIG
894               RD(K)=CRANG(0.0)
895               IF (K.LE.IDBG14) WRITE (6,100) NUMSAM,K,RD(K)
896       100     FORMAT(15H SAMPLE NUMBER=,I5,5X,15HVOLTAGE NUMBER=,I5,5X,
                *  8HVOLTAGE=,E15.8,2X,E15.8)
898       1     CONTINUE
899       C     COMPUTE SAMPLE
900             DO 2 I=1,DIM
901       C       GET RANDOM STARTING POINT TO SIMULATE RECEIVER NOISE
902               SUM=SQNOS(I)*CRANG(0.0)
903               ISUB=I+1
904               IF (ISUB.GT.DIM) ISUB=1
905               DO 3 K=1,NMEIG
906                 SUM=SUM+SQEIG(K)*RD(K)*SAVEXP(ISUB)
907                 ISUB=ISUB+I
908                 IF (ISUB.GT.DIM) ISUB=ISUB-DIM
909       3       CONTINUE
910               SMPR(I)=REAL(SUM)
911               SMPJ(I)=AIMAG(SUM)
912       2     CONTINUE
913             RETURN
914             ENTRY SAMPI
915       C     PRECALCULATE SQUARE ROOTS OF EIGENVALUE AND NOISE ARRAYS
916             SQNOS(1:DIM)=SQRT(NOISE(1:DIM))
917             SQEIG(1:NMEIG)=SQRT(EIG(1:NMEIG))
918             RETURN
919             END
```

```
920             COMPLEX FUNCTION CRANG(XS)

921       C     GENERATE A COMPLEX GAUSSIAN RANDOM NUMBER
922       C     IF (XS.GT.Ø)
923       C       XS IS NEW SEED
924       C       CRANG COMPUTED FROM XS AND NEXT NUMBER IN SEQUENCE
925       C     OR IF (XS.EQ.Ø)
926       C       CRANG COMPUTED FROM NEXT TWO NUMBERS IN SEQUENCE
927       C     OR IF (XS.LT.Ø)
928       C       CRANG IS LAST VALUE
929       C     END IF

930             DIMENSION RHO(4Ø97),ETHETA(1Ø26)
931             COMPLEX CRANGI
932             COMPLEX SAVRND
933             DATA MULT/442Ø564Ø89229/,X3/.2730509382710764Ø436/
934             DATA IB6/-47/,NX2/1/
935             DATA PI/3.14159265358979323/
936             IF (XS) 1,2,3
937       1     CONTINUE
938             CRANG=SAVRND
939             RETURN
940       2     CONTINUE
941             CALL Q8MPYL(MULT,X3,DX7)
942             CALL Q8PACK(IB6,DX7,R1)
943             X3=DX7

944       C                                                          )Ø.Ø(FNAR=

945             CALL Q8MPYL(MULT,X3,DX7)
946             CALL Q8PACK(IB6,DX7,R2)
947             X3=DX7

948       C                                                          )Ø.Ø(FNAR=

949             GOTO 4
950       3     CONTINUE
951             CALL Q8RIOR(NX2,XS,X3)
952             CALL Q8PACK(IB6,X3,R1)

953       C                                                           )SX(FNAR=

954             CALL Q8MPYL(MULT,X3,DX7)
955             CALL Q8PACK(IB6,DX7,R2)
956             X3=DX7

957       C                                                          )Ø.Ø(FNAR=

958       4     CONTINUE

959       C     TABLE LOOK UP FOR ABSOLUTE VALUE OF CRANG

960             JRHO=INT(R1*4Ø96.+1.)
961             RO=RHO(JRHO)

962       C     TABLE LOOK UP FOR ANGULAR PART OF CRANG
```

```
963             JTHETA=INT(R2*4096.+2.)
964             IF (JTHETA.LE.2049) GOTO 5
965               IF (JTHETA.LE.3073) GOTO 6
966       C         4TH QUADRANT
967                 JTRL=JTHETA-3072
968                 TRL=ETHETA(JTRL)
969                 JTIM=4099-JTHETA
970                 TIM=-ETHETA(JTIM)
971                 GOTO 7
972       6       CONTINUE
973       C         3RD QUADRANT
974                 JTRL=3075-JTHETA
975                 TRL=-ETHETA(JTRL)
976                 JTIM=JTHETA-2048
977                 TIM=-ETHETA(JTIM)
978                 GOTO 7
979       5     CONTINUE
980             IF (JTHETA.LE.1025) GOTO 8
981       C         2ND QUADRANT
982                 JTRL=JTHETA-1024
983                 TRL=-ETHETA(JTRL)
984                 JTIM=2051-JTHETA
985                 TIM=ETHETA(JTIM)
986                 GOTO 7
987       8       CONTINUE
988       C         1ST QUADRANT
989                 JTRL=1027-JTHETA
990                 TRL=ETHETA(JTRL)
991                 TIM=ETHETA(JTHETA)
992       7     CONTINUE
993             CRANG=RO*CMPLX(TRL,TIM)
994             SAVRND=CRANG
995             RETURN
996             ENTRY CRANGI(X)
997             CRANG=CMPLX(0.,0.)
998       C     INITIALIZE RHO AND ETHETA FOR TABLE LOOK UP
999             XX=-1./8192.
1000            XADD=1./4096.
1001            DO 9 J=1,4096
1002              XX=XX+XADD
1003              RHO(J)=SQRT(-ALOG(XX))
1004      9     CONTINUE
1005            RHO(4097)=RHO(4096)
1006            XX=-PI/4096.
1007            XADD=PI/2048.
```

```
1003              DO 10 J=2,1025
1009                XX=XX+XADD
1010                ETHETA(J)=SIN(XX)
1011        10    CONTINUE
1012              ETHETA(1)=ETHETA(2)
1013              ETHETA(1026)=ETHETA(1025)
1014              RETURN
1015              END
```

```
1016              SUBROUTINE DISP(N,X,JPRLEN)

1017        C     DEBUG OUTPUT ROUTINE FOR MATRICES WHOSE LOWER HALF ONLY IS STORED
1018        C     JTH ROW OF OUTPUT CORRESPONDS TO JTH ROW OF ACTUAL MATRIX TO THE
1019        C     RIGHT OF THE DIAGONAL ELEMENT (SIGN OF IMAGINARY PART REVERSED
1020        C     FOR HERMITIAN MATRIX)

1021              DIMENSION X(1)
1022              WRITE (6,100) (J,J=1,10)
1023        100   FORMAT(4X,10I12)
1024              ISUB=1
1025              DO 1 J=1,JPRLEN
1026                ILST=ISUB+N-J
1027                WRITE (6,101) J,(X(K),K=ISUB,ILST)
1028        101     FORMAT(1X,I3,10E12.5,/,(4X,10E12.5))
1029                ISUB=ILST+1
1030        1     CONTINUE
1031              RETURN
1032              END
```

4201 Lexington Avenue North
Arden Hills, Minnesota 55112
612/482-2100

CONTROL DATA CORPORATION

March 31, 1977

Mr. James Demmel
Technology Service Corp.
Data Sciences Division
2811 Wilshire Blvd.
Santa Monica, California 90903

Dear Mr. Demmel,

I have been asked to reply to your letter of March 11, 1977 to Mr. Dennis Kuba.

First, in order to effect the Choleski decomposition for a complex matrix it is required to do $2/3\ N^3 + 0\ (N^2)$ arithmetic operations. If it is necessary to do it in 9.10^{-3} sec. (a few milliseconds) for a 200 x 200 matrix, one needs a computer capable of $2/3\ .8.10^6\ /\ 9.10^{-3}$ FLOPS = 600 megaflops. This is without any consideration of "overhead". There is no computer presently capable of such performance. Indeed not even the 4 pipe STAR 100C will be capable of such performance on a 200 x 200 matrix.

Let me start with one general comment. The number of cycles needed to perform the decomposition is a cubic polynomial. Most of the points you raise will affect only the linear term or weakly affect higher order terms.

As for your specific questions about the compiler, it is relatively naive compared to the optimizing FORTRAN compilers for IBM 360/370 or CDC 7600's. The best way to get the answers to your compiler questions is to get an assembly listing and register map by exercising the proper options on the FORTRAN card.

Some things you should check for are:

(1) Loads and stores degrade performance greatly. Descriptor loads can impede vector operations. Rewriting assembly code to get bottom load, top store code is often advantagous. With this type of code the load can be "hidden" behind branch time.

{2} Fill the register file as full as possible with descriptors and index variables eliminating the need for load/store.

{3} Explicit dyadicization of vector arithmetic cuts down compiler generated overhead.

{4} As for the back substitution phase, if you have very many RHS's, all known at once, one can vectorize along the number of RHS's and perform "simultaneous solution".

{5} Perhaps you could pack the new starting points in your dynamic inner loop descriptors since the lengths are invariant inside the loop.

The basic vector algorithm you used is the most efficient one I know of and hence I can be of no help there.

I should be in the Los Angeles area sometime in the next 60 days. Perhaps we could meet and discuss your problem more fully, especially the plans for the STAR100C.

I hope I have been of some assistance.

Regards,

Dr. M. J. Kascic
STAR Consulting Services
STAR Operations Division

MJK/dm

Technology Service Corporation
Data Sciences Division
2811 Wilshire Boulevard, Santa Monica, California 90403 Phone: (213) 829-7411

6 April 1977

Dr. M. J. Kascic, Jr.
Consultant
STAR Operations Division
4290 Fernwood Avenue
St. Paul, Minnesota 55112

Dear Dr. Kascic:

Your letter discussing our algorithm for the Choleski decomposition of a positive definite Hermitian matrix arrived yesterday, and I wish to thank you for your time and effort. Your comments were generally clear and helpful, but I do have a question about your third numbered comment:

> Explicit dyadicization of vector arithmetic cuts down compiler generated overhead.

Do you mean to use expanded expressions like

```
DTEMP1 = DA*DB
DTEMP2 = DC*DD
DSUM   = DTEMP1 + DTEMP2
```

instead of

```
DSUM   = DA*DB + DC*DD
```

(where all variables are descriptors), or to use explicit vector references like SUM (1;LEN) instead of assigning a descriptor and using the descriptor?

We would be very happy to have you call or come in to TSC to talk to us when you are in Los Angeles. We look forward to discussing plans for the STAR 100 C.

Yours truly,

James Demmel

JD:cs

4201 Lexington Avenue North
Arden Hills, Minnesota 55112
612/482-2100

CONTROL DATA CORPORATION

April 26, 1977

Mr. James Demmel
Technological Service Corp.
2811 Wilshire Blvd.
Santa Monica, CA. 90403

Dear Mr. Demmel,

I am pleased that I have been of help. To answer your question about temporaries, a statement such as:

```
DSUM = DA*DB + DC*DD
```

should be expanded to:

```
DTEMP = DA*DB
DSUM = DC*DD
DSUM = DSUM + DTEMP
```

This forces only one temporary to be constructed. In general, gives an even more complicated set of algebraic statements, there is usually and "irreducible" number of temporaries needed. The compiler does not recognize this fact. Indeed the statement:

```
DC = DC + DA + DB
```

will compile into:

```
TEMP = DA + DB
DC = DC + TEMP
```

Clearly it is better to have:

```
DC = DC + DA
DC = DC + DB
```

Please let me know if I can be of any further assistance.

Regards,

CONTROL DATA CORPORATION

Dr. M. J. Kascic
STAR Consulting Services

MJK/dm

Appendix D. CRAY-1 Software

SUMMARY AND DOCUMENTATION

As discussed in Section 4.3.3.2, the algorithm and implementation chosen for the CRAY-1 are essentially the same as those chosen for the CDC STAR-100. Hence the documentation of the STAR is essentially correct for the CRAY except for one thing: the CRAY supports no explicit vectorization in its FORTRAN. The compiler, instead, examines DO loops in the program for suitability for vectorization. The program reads like standard FORTRAN. We have indicated where these vectorizable loops are in the program with comment statements.

The system routines used for timings are also different from the one on the STAR, and are called SECOND (CPU time) and RTC (real time).

```
      SUBROUTINE COVCMP
C     UPDATE SAMPLE COVARIANCE MATRIX
      COMMON /WORK/ MATDIM,MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
     *  RS,JS,REAL(20100),JMAG(20100),R(20100),C(20100),EREAL(200),
     *  EMAG(200),RECIP(200),TEMPR(200),TEMPJ(200)
      INTEGER SIZEM1
      REAL JS,JMAG
      COMMON /TIME/ TDECMP,TBKSB1,TBKSB2,TCVCMP,WDECMP,WBKSB1,WBKSB2,
     *  WCVCMP,TSETUP,WSETUP
      INTEGER CSTR,CEND,SSUB
      LOP=200/MATDIM
      W1=RTC(DUM)
      CALL SECOND(T1)
      DO 1 J=1,LOP
1     CONTINUE
      CALL SECOND(T2)
      W2=RTC(DUM)
      CALL SECOND(T3)
      DO 2 J=1,LOP
        CSTR=1
        MTDMMI=MTDMM1
        CEND=MATDIM
        SSUB=0
        DO 3 I=1,MATDIM
          SR=TEMPR(I)
          SJ=TEMPJ(I)
C*****    VECTOR OPERATIONS
          DO 4 JVEC=CSTR,CEND
            R(JVEC)=R(JVEC)+SR*TEMPR(JVEC-SSUB)+SJ*TEMPJ(JVEC-SSUB)
            C(JVEC)=C(JVEC)-SR*TEMPJ(JVEC-SSUB)+SJ*TEMPR(JVEC-SSUB)
4         CONTINUE
          CSTR=CEND+1
          CEND=CEND+MTDMMI
          SSUB=CEND-MATDIM
          MTDMMI=MTDMMI-1
3       CONTINUE
2     CONTINUE
      CALL SECOND(T4)
      W3=RTC(DUM)
C     CALCULATE CPU AND WALL CLOCK TIMES IN MILLISECONDS
      TCVCMP=(T4-T3-T2+T1)*1000./FLOAT(LOP)
      WCVCMP=(W3-2.*W2+W1)*12.5E-6/FLOAT(LOP)
      RETURN
      END
```

```
45              SUBROUTINE CHOLDC

46        C
47        C     PERFORM CHOLESKY LL* DECOMPOSITION OF A POSITIVE DEFINITE
48        C     HERMITIAN MATRIX
49        C

50              COMMON /WORK/ MATDIM,MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
                *  RS,JS,REEL(20100),JMAG(20100),R(20100),C(20100),EREAL(200),
                *  EMAG(200),RECIP(200),TEMPR(200),TEMPJ(200)
53              INTEGER SIZEM1
54              REAL JS,JMAG
55              COMMON /TIME/ TDECMP,TBKSB1,TBKSB2,TCVCMP,WDECMP,WBKSB1,WBKSB2,
                *  WCVCMP,TSETUP,WSETUP
57              INTEGER SSUB
58              INTEGER SUBCOL
59              W1=RTC(DUM)
60              CALL SECOND(T1)

61        C
62        C     SET UP ARRAY DESCRIPTORS
63        C

64              IF (MATDIM.EQ.MATOLD) GOTO 1
65                MATOLD=MATDIM
66                MTDMM1=MATDIM-1
67                MTDMM2=MATDIM-2
68                MTDMP1=MATDIM+1
69                MTDMP2=MATDIM+2
70                SIZEM1=MATDIM*(MATDIM+1)/2-1
71                CALL SECOND(T2)
72                W2=RTC(DUM)

73        C         CALCULATE CPU AND WALL CLOCK TIME IN MILLISECONDS

74                TSETUP=(T2-T1)*1000.
75                WSETUP=(W2-W1)*12.5E-6
76                RETURN
77        1     CONTINUE

78        C
79        C     CALCULATE RECIPROCAL OF FIRST DIAGONAL ELEMENT
80        C

81              IF (REEL(1).GT.0.0) GOTO 10
82                NONE=1
83                WRITE (6,500) NONE,REEL(1)
84        500     FORMAT(16H0ERROR IN CHOLDC,I4,18H DIAGONAL ELEMENT=,E12.5)
85                RETURN
86        10    CONTINUE
87              XTEMP=1./SQRT(REEL(1))
88              RECIP(1)=XTEMP

89        C
90        C     CALCULATE FIRST COLUMN
91        C
```

```
 92      C*****VECTOR OPERATIONS

 93            DO 1001 JVEC=1,MATDIM
 94              REEL(JVEC)=REEL(JVEC)*XTEMP
 95              JMAG(JVEC)=JMAG(JVEC)*XTEMP
 96      1001  CONTINUE

 97      C
 98      C     CALCULATE COLUMNS 2 THROUGH MATDIM-1
 99      C

100           ILAST=MATDIM
101           LENM1=MTDMM1
102           JCOLM1=0
103           DO 3 JCOL=2,MTDMM1
104             IFIRST=ILAST+1
105             ILAST=ILAST+LENM1
106             LENM1=LENM1-1
107             ISUB=JCOL
108             JCOLM1=JCOLM1+1

109     C
110     C       SUBTRACT MULTIPLES OF PREVIOUS COLUMNS
111     C

112             DO 4 SUBCOL=1,JCOLM1
113               TR=-REEL(ISUB)
114               TJ=JMAG(ISUB)
115               SSUB=IFIRST-ISUB

116     C*****    VECTOR OPERATIONS

117               DO 1002 JVEC=IFIRST,ILAST
118                 REEL(JVEC)=REEL(JVEC)+TR*REEL(JVEC-SSUB)-TJ*JMAG(JVEC-SSUB)
119                 JMAG(JVEC)=JMAG(JVEC)+TJ*REEL(JVEC-SSUB)+TR*JMAG(JVEC-SSUB)
120     1002      CONTINUE
121               ISUB=ISUB+MATDIM-SUBCOL
122     4       CONTINUE

123     C
124     C       CALCULATE RECIPROCAL OF DIAGONAL ELEMENT
125     C

126             IF (REEL(IFIRST).GT.0.0) GOTO 11
127               WRITE (6,500) JCOL,REEL(IFIRST)
128               RETURN
129     11      CONTINUE
130             XTEMP=1./SQRT(REEL(IFIRST))
131             RECIP(JCOL)=XTEMP

132     C
133     C       CALCULATE COLUMN
134     C
135     C*****  VECTOR OPERATIONS

136             DO 1003 JVEC=IFIRST,ILAST
```

```
137                 REEL(JVEC)=REEL(JVEC)*XTEMP
138                 JMAG(JVEC)=JMAG(JVEC)*XTEMP
139       1003    CONTINUE
140       3       CONTINUE

141       C
142       C       CALCULATE LAST DIAGONAL ELEMENT
143       C

144               SUM=0.
145               ISUB=MATDIM
146               DO 7 SUBCOL=1,MTDMM1
147                 XT=REEL(ISUB)
148                 XI=JMAG(ISUB)
149                 SUM=SUM+XT*XT+XI*XI
150                 ISUB=ISUB+MATDIM-SUBCOL
151       7       CONTINUE
152               ASUM=REEL(ISUB)-SUM
153               IF (ASUM.GT.0.0) GOTO 12
154                 WRITE (6,500) MATDIM,ASUM
155                 RETURN
156       12      CONTINUE
157               RECIP(MATDIM)=1./SQRT(ASUM)
158               RS=REEL(SIZEM1)
159               JS=JMAG(SIZEM1)
160               CALL SECOND(T2)
161               W2=RTC(DUM)

162       C       CALCULATE CPU AND WALL CLOCK WIMES IN MILLISECONDS

163               TDECMP=(T2-T1)*1000.
164               WDECMP=(W2-W1)*12.5E-6
165               RETURN
166               END
```

```
167             SUBROUTINE BAKSUB

168       C
169       C     GIVEN THE LL* DECOMPOSITION OF A MATRIX, AND A VECTOR S, PERFORM
170       C     TWO BACK SUBSTITUTIONS TO SOLVE (LL*)W=S FOR W
171       C

172             COMMON /WORK/ MATDIM,MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
                *  RS,JS,REEL(20100),JMAG(20100),R(20100),C(20100),EREAL(200),
                *  EMAG(200),RECIP(200),TEMPR(200),TEMPJ(200)
175             INTEGER SIZEM1
176             REAL JS,JMAG
177             COMMON /TIME/ TDECMP,TBKSB1,TBKSB2,TCVCMP,WDECMP,WBKSB1,WBKSB2,
                *  WCVCMP,TSETUP,WSETUP
179             INTEGER SADD
180             W1=RTC(DUM)
181             CALL SECOND(T1)

182       C
183       C     PERFORM FIRST BACK SUBSTITUTION, (L)TEMP=S
184       C
185       C     CALCULATE FIRST ELEMENT OF TEMP
186       C

187             XMUL=RECIP(1)
188             TEMPR(1)=EREAL(1)*XMUL
189             TEMPJ(1)=EMAG(1)*XMUL

190       C
191       C     SUBTRACT MULTIPLE OF FIRST COLUMN OF L FROM S AND STORE IN TEMP
192       C
193       C*****VECTOR OPERATIONS

194             DO 1001 JVEC=2,MATDIM
195               TEMPR(JVEC)=EREAL(JVEC)-TEMPR(1)*REEL(JVEC)+TEMPJ(1)*JMAG(JVEC)
196               TEMPJ(JVEC)=EMAG(JVEC)-TEMPJ(1)*REEL(JVEC)-TEMPR(1)*JMAG(JVEC)
197       1001  CONTINUE

198       C
199       C     CALCULATE ELEMENTS 2 THROUGH MATDIM-2
200       C

201             SADD=MTDMM1
202             DO 1 JCOL=2,MTDMM2

203       C
204       C       CALCULATE JCOL-TH ELEMENT
205       C

206               JCOLP1=JCOL+1
207               XMUL=RECIP(JCOL)
208               TEMPR(JCOL)=TEMPR(JCOL)*XMUL
209               TEMPJ(JCOL)=TEMPJ(JCOL)*XMUL

210       C
211       C       SUBTRACT MULTIPLE OF JCOL-TH COLUMN FROM TEMP
```

```
212       C
213                 TR=-TEMPR(JCOL)
214                 TJ=TEMPJ(JCOL)
215       C*****    VECTOR OPERATIONS
216                 DO 1002 JVEC=JCOLP1,MATDIM
217                   TEMPR(JVEC)=TEMPR(JVEC)+TR*REEL(JVEC+SADD)+TJ*JMAG(JVEC+SADD)
218                   TEMPJ(JVEC)=TEMPJ(JVEC)-TJ*REEL(JVEC+SADD)+TR*JMAG(JVEC+SADD)
219       1002      CONTINUE
220                 SADD=SADD+MATDIM-JCOL
221       1       CONTINUE
222       C
223       C       CALCULATE ELEMENT MATDIM-1
224       C
225                 XMUL=RECIP(MTDMM1)
226                 TEMPR(MTDMM1)=TEMPR(MTDMM1)*XMUL
227                 TEMPJ(MTDMM1)=TEMPJ(MTDMM1)*XMUL
228       C
229       C       COMBINE LAST STEP OF FIRST BACK SUBSTITUTION
230       C       (CALCULATION OF ELEMENT MATDIM) WITH FIRST STEP OF SECOND
231       C       BACK SUBSTITUTION (CALCULATION OF ELEMENT MATDIM OF W WHERE
232       C       (L*)*W=TEMP AND W IS STORED IN TEMP)
233       C
234                 XMUL=RECIP(MATDIM)**2
235                 TR1=TEMPR(MTDMM1)
236                 TJ1=TEMPJ(MTDMM1)
237                 TEMPR(MATDIM)=(TEMPR(MATDIM)-TR1*RS+TJ1*JS)*XMUL
238                 TEMPJ(MATDIM)=(TEMPJ(MATDIM)-TJ1*RS-TR1*JS)*XMUL
239       C
240       C       PERFORM SECOND BACK SUBSTITUTION, (L*)W=TEMP, STORING W IN TEMP
241       C
242                 CALL SECOND(T2)
243                 W2=RTC(DUM)
244                 CALL SECOND(T3)
245       C
246       C       CALCULATE MATDIM-1-ST ELEMENT
247       C
248                 XMUL=RECIP(MTDMM1)
249                 TR=TEMPR(MATDIM)
250                 TJ=TEMPJ(MATDIM)
251                 TEMPR(MTDMM1)=(TR1-TR*RS-TJ*JS)*XMUL
252                 TEMPJ(MTDMM1)=(TJ1-TJ*RS+TR*JS)*XMUL
253       C
254       C       CALCULATE ELEMENTS MATDIM-2 THROUGH 1
255       C
```

```
256             SADD=SIZEM1-MATDIM
257             JROW=MTDMM1
258             DO 2 JTEMP=2,MTDMM1
259               JROWP1=JROW
260               JROW=JROW-1
261               SADD=SADD-JTEMP

262       C
263       C       CALCULATE DOT PREDUCT OF JROW-TH ROW STARTING AT COLUMN JROW+1
264       C       WITH TEMP STARTING WITH THE JROW+1-ST ELEMENT
265       C
266       C*****  VECTOR OPERATIONS, DOT PRODUCTS

267               DOTR=0.
268               DOTJ=0.
269               DO 1003 JVEC=JROWP1,MATDIM
270                 DOTR=DOTR+TEMPR(JVEC)*REEL(JVEC+SADD)+TEMPJ(JVEC)*
                *     JMAG(JVEC+SADD)
272                 DOTJ=DOTJ+TEMPJ(JVEC)*REEL(JVEC+SADD)-TEMPR(JVEC)*
                *     JMAG(JVEC+SADD)
274       1003      CONTINUE

275       C
276       C       CALCULATE JROW-TH ELEMENT
277       C

278               XMUL=RECIP(JROW)
279               TEMPR(JROW)=(TEMPR(JROW)-DOTR)*XMUL
280               TEMPJ(JROW)=(TEMPJ(JROW)-DOTJ)*XMUL
281       2       CONTINUE
282               CALL SECOND(T4)
283               W3=RTC(DUM)

284       C       CALCULATE CPU AND WALL CLOCK TIMES IN MILLISECONDS

285               TBKSB1=(T2-T1)*1000.
286               TBKSB2=(T4-T3)*1000.
287               WBKSB1=(W2-W1)*12.5E-6
288               WBKSB2=(W3-W2)*12.5E-6
289               RETURN
290               END
```

```
291             PROGRAM SYSTST(TAPE6=OUTPUT)

292       C     TEST ADAPTIVE ARRAY SYSTEM ON CRAY-1
293       C      VECTOR OPERATIONS ARE INDICATED BY COMMENT CARDS WITH
294       C     'C*****' IN COLUMNS 1 THROUGH 6

295             COMMON /WORK/ MATDIM,MATOLD,MTDMM1,MTDMM2,MTDMP1,MTDMP2,SIZEM1,
                *  RS,JS,REEL(20100),JMAG(20100),R(20100),C(20100),EREAL(200),
                *  EMAG(200),RECIP(200),TEMPR(200),TEMPJ(200)
298             INTEGER SIZEM1
299             REAL JS,JMAG
300             COMMON /TIME/ TDECMP,TBKSB1,TBKSD2,TCVCMP,WDECMP,WBKSB1,WBKSB2,
                *  WCVCMP,TSETUP,WSETUP
302             DATA EREAL/200*1./,EMAG/200*1./
303             MATDIM=0
304             WRITE (6,100)
305       100   FORMAT(41H1     NUMBER  TIME FOR  TIME FOR  TIME FOR,
                *50H  TIME FOR    TIME TO   TIME TO   TIME TO MAX TOTAL,/,
                *51H    SAMPLES     SET UP    DECOMP BACKSUB 1 BACKSUB 2,
                *40H       MOVE  ZERO OUT    UPDATE       TIME)
309       1     CONTINUE

310       C     GET NEXT DIMENSION

311             MATDIM=NEXDIM(MATDIM)
312             IF (MATDIM.EQ.0) STOP 777

313       C     TEST REINITIALIZATION

314             LEN=MATDIM*(MATDIM+1)/2
315             W1=RTC(DUM)
316             CALL SECOND(T1)

317       C*****VECTOR OPERATION

318             DO 2 JVEC=1,LEN
319               REEL(JVEC)=1.
320               JMAG(JVEC)=1.
321       2     CONTINUE
322             CALL SECOND(T2)
323             W2=RTC(DUM)

324       C     CALCULATE CPU AND WALL CLOCK TIMES IN MILLISECONDS

325             TZERO=(T2-T1)*1000.
326             WZERO=(W2-W1)*12.5E-6

327       C     TEST MOVING VECTORS

328             W1=RTC(DUM)
329             CALL SECOND(T1)

330       C*****VECTOR OPERATIONS

331             DO 3 JVEC=1,LEN
332               R(JVEC)=REEL(JVEC)
```

```
333                 C(JVEC)=JMAG(JVEC)
334           3     CONTINUE
335                 CALL SECOND(T2)
336                 W2=RTC(DUM)
337           C     CALCULATE CPU AND WALL CLOCK TIMES IN MILLISECONDS
338                 TMOVE=(T2-T1)*1000.
339                 WMOVE=(W2-W1)*12.5E-6
340           C     TEST SET UP OF POINTERS
341                 MATOLD=MATDIM-1
342                 CALL CHOLDC
343           C     FILL SAMPLE VECTORS
344           C*****VECTOR OPERATIONS
345                 DO 5 JVEC=1,MATDIM
346                   TEMPR(JVEC)=1.
347                   TEMPJ(JVEC)=1.
348           5     CONTINUE
349           C     TEST UPDATE OF COVARIANCE MATRIX
350                 CALL COVCMP
351           C     FILL UP MATRIX WITH POSITIVE DEFINITE HERMITIAN DATA
352                 ISUB=1
353                 DUM=10.*FLOAT(MATDIM)
354                 DO 4 J=1,MATDIM
355                   REEL(ISUB)=DUM
356                   JMAG(ISUB)=0.
357                   ISUB=ISUB+MATDIM-J+1
358           4     CONTINUE
359           C     TEST DECOMPOSITION
360                 CALL CHOLDC
361           C     TEST BACK SUBSTITUTIONS
362                 CALL BAKSUB
363           C     CALCULATE TIMES
364                 TMAX=TSETUP+TDECMP+TBKSB1+TBKSB2+TMOVE+TZERO+2.*FLOAT(MATDIM)*
                  *TCVCMP
365                 WMAX=WSETUP+WDECMP+WBKSB1+WBKSB2+WMOVE+WZERO+2.*FLOAT(MATDIM)*
                  *WCVCMP
368                 WRITE (6,101) MATDIM,TSETUP,TDECMP,TBKSB1,TBKSB2,TMOVE,TZERO,
                  *TCVCMP,TMAX,WSETUP,WDECMP,WBKSB1,WBKSB2,WMOVE,WZERO,WCVCMP,WMAX
370           101   FORMAT(1H0,I10,8F10.3,5X,13HCPU(MILLISEC),/,11X,8F10.3,5X,
                  *14HWALL(MILLISEC))
372                 GOTO 1
373                 END
```

```
374             FUNCTION NEXDIM(N)
375       C     GENERATE DIMENSIONS TO DRIVE SYSTEM
376             IF (N.GT.Ø) GOTO 1
377             NEXDIM=2Ø
378             RETURN
379       1     CONTINUE
38Ø             IF (N.GE.5Ø) GOTO 2
381             NEXDIM=N+5
382             RETURN
383       2     CONTINUE
384             IF (N.GE.2ØØ) GOTO 3
385             NEXDIM=N+1Ø
386             RETURN
387       3     CONTINUE
388             NEXDIM=Ø
389             RETURN
39Ø             END
```

Appendix E. Complex Multiplication

The problem is to multiply two complex numbers a+bi and c+di to obtain the product $p_r+p_i i$ in the "best" way.

"Best" is determined not only by the operations count, but depends on

1) the relative times it takes the machine to do an addition (+) and a multiplication(*)

2) the computer architecture (vector processors, like the CDC STAR, parallel processors like the ILLIAC IV, machines like the CDC 7600 which overlap execution of instructions, and any unmentioned or as-yet-unbuilt combination of the above)

3) number of complex multiplications to be performed (since algorithms require different amounts of temporary storage, which may be large on a vector machine).

All these considerations are machine- and problem-dependent. In fact, the third one could result in one program using two different algorithms, if different numbers of complex numbers are to be multiplied at different times.

Just on the basis of operation counts, the following two methods seem best (where a+bi and c+di are the multiplicands and $p_r+p_i i$ is the product):

1) Assuming an identifier can appear on only one side of the equal sign:

1.1 $t_1 = a*c$

1.2 $t_2 = b*d$

1.3 $p_r = t_1 - t_2$

1.4 $t_1 = b*c$

1.5 $t_2 = a*d$

1.6 $p_i = t_1 + t_2$

multiplies = 4

additions and subtractions = 2

temporaries = 2.

Assuming an identifier can appear on both sides of the equal sign:

2.1 $p_r = a*c$

2.2 $t_1 = b*d$

2.3 $p_r = p_r + t_1$

2.4 $p_i = b*c$

2.5 $t_1 = a*d$

2.6 $p_i = p_i + t_1$

multiplies = 4

additions and subtractions = 2

temporaries = 2.

2) Assuming an identifier can appear on only one side of the equal sign:

3.1 $t_1 = a + b$

3.2 $t_2 = c - d$

3.3 $t_3 = t_1*t_2$

3.4 $t_1 = a*d$

3.5 $t_2 = b*c$

3.6 $p_i = t_1 + t_2$

3.7 $t_4 = t_3 - t_2$

3.8 $p_r = t_4 + t_1$

} or $\begin{cases} p_r + t_3 - t_2 \\ t_3 = p_r + t_1 \\ p_r = t_3 \end{cases}$

multiplies = 3

additions and subtractions = 5

temporaries = 4 or 3 .

3) Assuming an identifier can appear on both sides of the equal sign:

4.1 $p_r = a + b$

4.2 $p_i = c - d$

4.3 $t_i = p_i * p_r$

4.4 $t_2 = a * d$

4.5 $p_r = b * c$

4.6 $p_i = t_2 + p_r$

4.7 $p_r = t_1 - p_r$

4.8 $p_r = p_i + t_2$

multiplies = 3

additions and subtractions = 5

temporaries = 2.

Some vector machines may not allow a vector identifier to appear on both sides of the equal sign, which is the reason for two separate implementations of the same algorithm. Also, the number of temporaries is of interest to people using assembly language or an optimizing compiler which tries to keep all temporaries in registers, of which there are only a limited number. A user using a machine which overlaps instructions may wish to use more temporaries and allow more time between an instruction which uses a result of a previous instruction in order to allow an instruction

to finish. Algorithm 1, for example, might look like this on a CDC 7600:

$$1.1' \quad t_1 = a*c$$

$$1.2' \quad t_2 = b*d$$

$$1.3' \quad p_r = t_1 - t_2$$

$$1.4' \quad t_3 = a*d$$

$$1.5' \quad t_4 = b*c$$

$$1.6' \quad p_i = t_3 + t_4.$$

This implementation uses two more temporaries but allows the multiply units to begin work in line 1.4' before the addition unit is done in line 1.3'. Algorithm 2 could change to

$$3.1' \quad t_1 = a*d$$

$$3.2' \quad t_2 = a + b$$

$$3.3' \quad t_3 = b*c$$

$$3.4' \quad t_4 = c - d$$

$$3.5' \quad t_5 = t_2*t_4$$

$$3.6' \quad p_r = t_1 + t_3$$

$$3.7' \quad t_2 = t_5 - t_3$$

$$3.8' \quad p_i = t_2 + t_1$$

where 5 temporaries are used instead of 4. Here there are only 3 instructions which must wait for the previous instruction to finish, whereas in the original implementation there were 7.

To compare these two algorithms from an operation counts point of view, let t_a be the number of seconds required to perform an addition or

subtraction and t_m be the time required to perform a multiplication. Algorithm 1 requires $4t_m + 2t_a$ seconds and Algorithm 2 requires $3t_m + 5t_a$ seconds, so Algorithm 1 is faster when $t_m < 3t_a$ and Algorithm 2 is faster when $t_m > 3t_a$.

To show how the analysis changes from machine to machine, we consider a parallel processor with 4 independent, communicating processors, each of which performs an operation and waits for the others to finish. In this case Algorithm 1 becomes

1) $t_1 = a*c \quad t_2 = b*d \quad t_3 = b*c \quad t_4 = a*d$ (simultaneous)

2) $p_r = t_1 - t_2 \qquad p_i = t_3 + t_4$ (simultaneous)

for a total time of $t_a + t_m$ with 4 temporaries; whereas, Algorithm 2 becomes

1) $t_1 = a*d \quad t_2 = b*c \quad t_3 = a+b \quad t_4 = c-d$ (simultaneous)

2) $p_i = t_1 + t_2 \quad t_5 = t_3*t_4 \quad t_6 = t_1 - t_2$ (simultaneous)

3) $p_r = t_5 + t_6$

for a total time of $2\ t_m + t_a$ with 6 temporaries, obviously more than Algorithm 1.

It can be shown that the multiplication of 2 complex numbers requires at least 3 multiplications;* but we know of no results for the minimum number of additions. The two algorithms mentioned above, however, are the best we have found.

On the computers of interest, the four-multiply-two-add method is always faster except for integers on the STARAN. The only add and multiply times we have now are for 16-bit numbers. The current STARAN has timings of

*Aho, A.V., J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, New York, 1974, Chapter 12.

19 μsecs and 283 μsecs for add and multiply, respectively. The STARAN-E system has timings of 11.3 μsecs and 119 μsecs.

Appendix F. Implementations of Algorithms for Direct Methods of Solving $MW = \overline{S}$ on Sequential Processors

All the implementations are written in an easily understood structured programming language. These implementations are presented for operations count purposes and are not intended to be compilable code.

In the following discussion, sample covariance matrix will be abbreviated to SCM. MR and MI will be arrays containing, respectively, the real and imaginary parts of the SCM, and will be floating-point (FP). SR and SI are FP arrays containing the real and imaginary parts of the steering vector or vectors. XR and XI will contain the real and imaginary parts of the sample voltage vectors. Preceding each algorithm will be a list of variables with their types (floating-point (FP) or integer (I)), their lengths (as a function of the number of weights (N), the number of samples (N_S) and the number of steering vectors (K)), and contents (omitted for MR, MI, XR, XI, SR, SI; contents include storage scheme (rowwise, columnwise, etc.)).

FOR loops behave as they should: if the initial value is greater than the final value, and the increment (default = 1) is positive, or if the initial value is less than the final value and the increment is negative, the loop is bypassed.

TABLE OF CONTENTS

SCM Calculation

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
XR	FP	N	
XI	FP	N	
ROW	I	1	current row of SCM
START	I	1	location of current element of SCM
COLUMN	I	1	current column of SCM

```
      BEGIN
      START = 1

C     UPDATE EACH ROW

      FOR ROW  = 1 TO N
     .   MR(START)   = MR(START)   + XR(ROW)*XR(ROW)
     +.                +XI(ROW)*XI(ROW)
     .   UPDATE EACH ELEMENT OF CURRENT ROW
     .   FOR COLUMN = ROW + 1 TO N
     .   .   START =  START + 1
     .   .   MR(START) = MR(START) + XR(ROW)*XR(COLUMN)
     +.  .               +XI(ROW)*XI(COLUMN)
     .   .   MI(START)   =  MI(START) + XR(ROW)*XI(COLUMN)
     +.  .               -XI(ROW)*XR(COLUMN)
     .   END FOR
      END FOR
END
```

Cholesky without Square Roots (LDL*) Decomposition

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
TR	FP	N	temporary array
TI	FP	N	temporary array
RECIP	FP	N	reciprocal of elements of D in decomposition M = LDL*
ROW	I	1	current row of SCM
SUBROW	I	1	row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROWth element of ROWth row (diagonal element)
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LENGTH	I	1	length of ROWth row starting at element ROW + 1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW + 1
LOC	I	1	starting location increment used to point to current element of SCM being calculated

```
      BEGIN
      STARTR = 1
      LENGTH = N

C     CALCULATE  NOWTH ROW OF L* FACTOR IN  M = LDL*

      FOR ROW = 1 TO N

C     *   MAKE UNIT DIAGONAL

      .   RECIP(ROW) = 1./MR(STARTR)
      .   LENGTH = LENGTH -1
      .   FOR  LOC =  1  TO LENGTH
      .   .   TR(ROW+LOC) = RECIP(ROW)*MR(STARTR+LOC)
      .   .   TI(ROW+LOC) = RECIP(ROW)*MI(STARTR+LOC)
      .   END FOR

C     *   SUBTRACT MULTIPLES OF NOWTH ROW FROM OTHER ROWS

      .   LENGTHS =  LENGTH  - 1
      .   STARTRS  = STARTR + 1
      .   STARTS =  STARTR + LENGTH + 1
      .   FOR SUBROW  = ROW + 1 TO N
      .   .   MR(STARTS) = MR(STARTS) - TR(SUBROW)*
     +.   .              MR(STARTRS) - TI(SUBROW)*
     +.   .              MI(STARTRS)
      .   .   FOR LOC = 1 TO LENGTHS
      .   .   .   MR(STARTS + LOC) = MR(STARTS +  LOC)
     +.   .   .                -TR(SUBROW)*MR(STARTRS+LOC)
     +.   .   .                -TI(SUBROW)*MI(STARTRS+LOC)
      .   .   .   MI(STARTS+LOC) =  MI(STARTS+LOC)
     +.   .   .                -TR(SUBROW)*MI(STARTRS+LOC)
     +.   .   .                +TI(SUBROW)*MR(STARTRS+LOC)
      .   .   END FOR
      .   .   STARTRS  =  STARTRS + 1
      .   .   STARTS = STARTS + LENGTHS + 1
      .   .   LENGTHS = LENGTHS -1
      .   END FOR

C     *   MOVE NORMALISED ROW FROM TR AND  TI  TO MR AND MI

      .   FOR LOC =  1 TO  LENGTH
      .   .   MR(STARTR+LOC) = TR(STARTR+LOC)
      .   .   MI(STARTR+LOC) = TI(STARTR+LOC)
      .   END FOR
      .   STARTR = STARTR + LENGTH + 1
      END FOR
      END
```

Cholesky with Square Roots (LL*) Decomposition

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	rowwise
MI	FP	N(N+1)/2	rowwise
RECIP	FP	N	reciprocals of diagonal elements of L in decomposition M = LL*
ROW	I	1	current row of SCM
SUBROW	I	1	Row from which multiple of ROWth row is subtracted
STARTR	I	1	location of ROWth element of ROWth row (diagonal elements)
STARTS	I	1	location of SUBROWth element of SUBROWth row
STARTRS	I	1	location of SUBROWth element of ROWth row
LENGTH	I	1	length of ROWth row starting at element ROW + 1
LENGTHS	I	1	length of SUBROWth row starting at element SUBROW + 1
LOC	I	1	starting location increment used to point to current element of SCM being calculated

```
      BEGIN
      STARTR = 1
      LENGTH = N

C     CALCULATE NEWTH ROW OF L* FACTOR IN M  = LL*

      FOR  ROW = 1 TO N
      .    RECIP(ROW) = 1./SQRT (MR(STARTR))
      .    LENGTH = LENGTH -1
      .    FOR LOC = 1 TO LENGTH
      .    .    MR(STARTR+LOC) = RECIP(ROW)*MR(STARTR+LOC)
      .    .    MI(STARTR+LOC)  = RECIP(ROW)*MI(STARTR+LOC)
      .    END FOR

C     *    SUBTRACT MULTIPLES OF NEWTH ROW FROM OTHER ROWS

      .    LENGTHS = LENGTH -1
      .    STARTRS = STARTR + 1
      .    STARTS = STARTR + LENGTH + 1
      .    FOR SUBROW = ROW + 1 TO N
      .    .    MR(STARTS) = MR(STARTS) -MR(STARTRS)*
     +.    .                MR(STARTRS)-MI(STARTRS)*MI(STARTRS)
      .    .    FOR LOC = 1 TO LENGTHS
      .    .    .    MR(STARTS+LOC) = MR(STARTS+LOC)
     +.    .    .                   -MR(STARTRS)*MR(STARTRS+LOC)
     +.    .    .                   -MI(STARTRS)*MI(STARTRS+LOC)
      .    .    .    MI(STARTS+LOC) = MI(STARTS+LOC)
     +.    .    .                   -MR(STARTRS)*MI(STARTRS+LOC)
     +.    .    .                   +MI(STARTRS)*MR(STARTRS+LOC)
      .    .    END FOR
      .    .    STARTRS = STARTRS + 1
      .    .    STARTS = STARTS + LENGTHS + 1
      .    .    LENGTHS = LENGTHS -1
      .    END FOR
      .    STARTR  = STARTR + LENGTH + 1
      END FOR
      END
```

First Back Substitution for LDL* (LDT = $\tilde{S}$)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	real part of L* factor rowwise
MI	FP	N(N+1)/2	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of elements of D factor
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location preceding first location of current steering vector
STARTSVA	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW + 1
STARTR	I	1	location of ROWth element of ROWth row of SCM
LOC	I	1	location increment used to address current element of steering vector being calculated

```
      BEGIN
      STARTSV = 0
C     DO EACH STEERING VECTOR
      FOR AUG = 1 TO K
C     *   SUBTRACT MULTIPLES OF EACH ROW OF M FROM THE STEERING VECTORS
      .   STARTSVA = STARTSV + 1
      .   STARTR = 1
      .   LENGTH  = N-1
      .   FOR  ROW = 1 TO N-1
      .   .   FOR LOC = 1 TO LENGTH
      .   .   .   SR(STARTSVA+LOC) = SR(STARTSVA+LOC)
     +.   .   .                  -SR(STARTSVA)*MR(STARTR+LOC)
     +.   .   .                  -SI(STARTSVA)*MI(STARTR+LOC)
      .   .   .   SI(STARTSVA+LOC) = SI(STARTSVA+LOC)
     +.   .   .                  +SR(STARTSVA)*MI(STARTR+LOC)
     +.   .   .                  -SI(STARTSVA)*MR(STARTR+LOC)
      .   .   END FOR
      .   .   STARTSVA =  STARTSVA + 1
      .   .   STARTR = STARTR + LENGTH + 1
      .   .   LENGTH = LENGTH -1
      .   END FOR
C     *   MULTIPLY BY RECIPROCAL DIAGONALS
      .   FOR LOC = 1 TO N
      .   .   SR(STARTSV+LOC) = SR(STARTSV+LOC)*RECIP(LOC)
      .   .   SI(STARTSV+LOC) =  SI(STARTSV+LOC)*RECIP(LOC)
      .   END FOR
      .   STARTSV =  STARTSV + N
      END FOR
      END
```

First Back Substitution for LL* (LT = $\bar{S}$)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	real parts of L* factor rowwise
MI	FP	N(N+1)/2	imaginary parts of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L factor
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location preceding first location of current steering vector
STARTSVA	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW + 1
STARTR	I	1	location of ROWth element of ROWth row of SCM
LOC	I	1	location increment used to address current element of steering vector being calculated

```
      BEGIN
      STARTSV = Ø

C     DO EACH STEERING VECTOR

      FOR AUG = 1 TO K

C     *   SUBTRACT MULTIPLES OF EACH ROW OF M FROM THE STEERING VECTORS

      .   STARTSVA = STARTSV +  1
      .   STARTR = 1
      .   LENGTH = N-1
      .   FOR ROW = 1 TO N
      .   .   SR(STARTSVA) = RECIP(ROW)*SR(STARTSVA)
      .   .   SI(STARTSVA) = RECIP(ROW)*SI(STARTSVA)
      .   .   FOR LOC = 1 TO LENGTH
      .   .   .   SR(STARTSVA +LOC) = SR(STARTSVA+LOC)
     +.   .   .                   -SR(STARTSVA)*MR(STARTR+LOC)
     +.   .   .                   -SI(STARTSVA)*MI(STARTR+LOC)
      .   .   .   SI(STARTSVA+LOC)  =  SI(STARTSVA+LOC)
     +.   .   .                   +SR(STARTSVA)*MI(STARTR+LOC)
     +.   .   .                   -SI(STARTSVA)*MR(STARTR+LOC)
      .   .   END FOR
      .   .   STARTSVA = STARTSVA+1
      .   .   STARTR = STARTR + LENGTH + 1
      .   .   LENGTH = LENGTH-1
      .   END FOR
      .   STARTSV =  STARTSV + N
      END FOR
      END
```

Second Back Substitution for LDL* (L*W = T)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	real part of L* factor rowwise
MI	FP	N(N+1)/2	imaginary part of L* factor rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
DOTR	FP	1	temporary location
DOTI	FP	1	temporary location
LAST	I	1	location of last element of SCM
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location of last element of current steering vector
STARTR	I	1	location of ROWth element of ROWth row of SCM
STARTSVS	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1
LOC	I	1	location increment used to address current element of SCM and steering vector

```
      BEGIN
      LAST = N*(N+1)/2
      STARTSV =  N

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO K
      .   STARTR =  LAST
      .   STARTSVS =  STARTSV-1
      .   LENGTH =1
      .   FOR ROW = N-1 TO 1  STEP -1
      .   .   STARTR = STARTR - LENGTH -1
      .   .   DOTR = Ø.
      .   .   DOTI = Ø.
      .   .   FOR  LOC  =  1 TO LENGTH
      .   .   .   DOTR  = DOTR + MR(STARTR+LOC)*SR(STARTSVS+LOC)
     +.   .   .           -MI(STARTR+LOC)*SR(STARTSVS+LOC)
      .   .   .   DOTI = DOTI + MR(STARTR+LOC)*
     +.   .   .                 SI(STARTSVS+LOC) + MI(STARTR+LOC)*SR(STARTSVS+LOC)
      .   .   END FOR
      .   .   SR(STARTSVS) =  SR(STARTSVS) - DOTR
      .   .   SI(STARTSVS) = SI(STARTSVS) - DOTI
      .   .   STARTSVS  =  STARTSVS-1
      .   .   LENGTH = LENGTH + 1
      .   END FOR
      .   STARTSV= STARTSV+N
      END FOR
      END
```

Second Back Substitution for LL* (L*W = T)

Variable	Type	Length	Contents
MR	FP	N(N+1)/2	real part of L* rowwise
MI	FP	N(N+1)/2	imaginary part of L* rowwise
SR	FP	KN	real part of steering vectors vectorwise
SI	FP	KN	imaginary part of steering vectors vectorwise
RECIP	FP	N	reciprocals of diagonal elements of L*
DOTR	FP	1	temporary location
DOTI	FP	1	temporary location
LAST	I	1	location of last element of SCM
AUG	I	1	current steering vector
ROW	I	1	current row of SCM
STARTSV	I	1	location of last element of current steering vector
STARTR	I	1	location of ROWth element of ROWth row of SCM
STARTSVS	I	1	location of ROWth element of current steering vector
LENGTH	I	1	length of ROWth row of SCM starting at element ROW+1
LOC	I	1	location increment used to address current element of SCM and steering vector

```
      BEGIN
      LAST = N*(N+1)/2
      STARTSV = N

C     ELIMINATE EACH STEERING VECTOR

      FOR AUG = 1 TO  K
      .   STARTR = LAST
      .   STARTSVS = STARTSV-1
      .   LENGTH = 1

C     *   CALCULATE N-TH ELEMENT

      .   SR(STARTSV) = SR(STARTSV)*RECIP(N)
      .   SI(STARTSV) = SI(STARTSV)*RECIP(N)

C     *   CALCULATE OTHER ELEMENTS

      .   FOR ROW = N-1  TO 1 STEP -1
      .   .   STARTR = STARTR - LENGTH -1
      .   .   DOTR = Ø.
      .   .   DOTI = Ø.
      .   .   FOR LOC = 1 TO LENGTH
      .   .   .   DOTR  = DOTR + MR(STARTR+LOC)*
     +.   .   .               SR(STARTSVS+LOC)  - MI(STARTR+LOC)*
     +.   .   .               SI(STARTSVS+LOC)
      .   .   .   DOTI = DOTI + MR(STARTR+LOC)*SI(STARTSVS+LOC) +
     +.   .   .               MI(STARTR+LOC)*SR(STARTSVS+LOC)
      .   .   END FOR
      .   .   SR(STARTSVS) = (SR(STARTSVS) -DOTR)*RECIP(ROW)
      .   .   SI(STARTSVS) =  (SI(STARTSVS) -DOTI)*RECIP(ROW)
      .   .   STARTSVS  = STARTSVS -1
      .   .   LENGTH = LENGTH + 1
      .   END FOR
      .   STARTSV = STARTSV  + N
      END FOR
      END
```

Appendix G. CDC 7600 Software

The algorithms and implementations of these programs are discussed in Section 4.2.3. Program COVTST and subroutines NEXDIM and SEKOND are drivers for COVCMP, which calculates the sample covariance matrix. DECTST, NEXDEC, and SEKOND drive CHOL, which performs the Cholesky decomposition and both back substitutions.

```
1             PROGRAM COVTST(OUTPUT,TAPE6=OUTPUT)

2       C     TEST COVARIANCE CALCULATER ON CDC7600 MINNEAPOLIS

3             COMMON /TIMING/ TZERO,TCVCMP,WZERO,WCVCMP
4             COMMON /TIME/ ICSEC,IRSEC
5             COMMON/SSTORE/YR(200),YC(200)
6             COMMON/INFO/N,R(20100),C(20100),BR(200),BC(200),WR(200),WC(200)
7             DATA YR/200*1.0/,YC/200*1.0/
8             WRITE (6,100)
9       100   FORMAT(1H1,6X,9HDIMENSION,7X,8HZERO OUT,7X,8H1 SAMPLE,5X,
             *  10H2N SAMPLES,10X,5HTOTAL)
11            N=0
12      999   CONTINUE

13      C     GET NEXT DIMENSION

14            N=NEXDIM(N)
15            IF (N.EQ.0) STOP 777

16      C     TEST ZERO OUT

17            LEN=N*(N+1)/2
18            LOP=2000/N
19            CALL SEC
20            IW1=IRSEC
21            IC1=ICSEC
22            DO 991 JLOP=1,LOP
23              DO 1 J=1,LEN
24                R(J)=0.
25                C(J)=0.
26      1       CONTINUE
27      991   CONTINUE
28            CALL SEC
29            IW2=IRSEC
30            IC2=ICSEC
31            DO 992 JLOP=1,LOP
32      992   CONTINUE
33            CALL SEC
34            IC3=IC2-IC1
35            IC4=ICSEC-IC2
36            IC5=IC3-IC4
37            IW3=IW2-IW1
38            IW4=IRSEC-IW2
39            IW5=IW3-IW4
40            TZERO=FLOAT(IC5)*27.5E-6/FLOAT(LOP)
41            WZERO=FLOAT(IW5)*27.5E-6/FLOAT(LOP)

42      C     TEST COVCMP

43            CALL COVCMP
44            W2N=2*N*WCVCMP
45            WTOT=WZERO+W2N
46            T2N=2*N*TCVCMP
47            TTOT=TZERO+T2N
48            WRITE (6,101) N,TZERO,TCVCMP,T2N,TTOT,WZERO,WCVCMP,W2N,WTOT
49      101   FORMAT(1H0,I15,4F15.3,5X,13HCPU(MILLISEC),/,16X,4F15.3,5X,
             *14HWALL(MILLISEC))
51            GOTO 999
52            END
```

```
53              SUBROUTINE COVCMP
54              COMMON /TIMING/ TZERO,TCVCMP,WZERO,WCVCMP
55              COMMON /TIME/ ICSEC,IRSEC
56              COMMON/SSTORE/YR(200),YC(200)
57              COMMON/INFO/N,R(20100),C(20100),BR(200),BC(200),WR(200),WC(200)
58              LOP=1000/N
59              CALL SEC
60              IW1=IRSEC
61              IC1=ICSEC
62              DO 991 JLOP=1,LOP
63              JM1=0
64              R(1)=YR(1)*YR(1)+YC(1)*YC(1)
65              INDEX=1
66              DO 1 J=2,N
67              JM1=JM1+1
68              SR=YR(J)
69              SC=YC(J)
70              DO 2 K=1,JM1
71              R(INDEX)=R(INDEX)+SR*YR(K)+SC*YC(K)
72              C(INDEX)=C(INDEX)+SR*YC(K)-SC*YR(K)
73              INDEX=INDEX+1
74      2       CONTINUE
75              R(INDEX)=R(INDEX)+SR*SR+SC*SC
76              INDEX=INDEX+1
77      1       CONTINUE
78      991     CONTINUE
79              CALL SEC
80              IW2=IRSEC
81              IC2=ICSEC
82              DO 992 JLOP=1,LOP
83      992     CONTINUE
84              CALL SEC
85              IC3=IC2-IC1
86              IC4=ICSEC-IC2
87              IC5=IC3-IC4
88              IW3=IW2-IW1
89              IW4=IRSEC-IW2
90              IW5=IW3-IW4
91              TCVCMP=FLOAT(IC5)*27.5E-6/FLOAT(LOP)
92              WCVCMP=FLOAT(IW5)*27.5E-6/FLOAT(LOP)
93              RETURN
94              END
```

```
 95              FUNCTION NEXDIM(N)
 96              IF (N.GT.Ø) GOTO 1
 97              NEXDIM=4
 98              RETURN
 99        1     IF (N.GE.3Ø) GOTO 2
1ØØ              NEXDIM=N+1
1Ø1              RETURN
1Ø2       2     IF (N.GE.5Ø) GOTO 3
1Ø3              NEXDIM=N+5
1Ø4              RETURN
1Ø5       3     IF (N.GE.2ØØ) GOTO 4
1Ø6              NEXDIM=N+1Ø
1Ø7              RETURN
1Ø8       4     NEXDIM=Ø
1Ø9              RETURN
11Ø              END
```

```
111                IDENT   SEKOND
112                USE     /TIME/
113       ICSEC    BSS     1
114       IRSEC    BSS     1
115                USE     *
116                ENTRY   SEC
117       SEC      BSS     1
118                RTIME   IRSEC,7CLK
119                TIME    ICSEC,7CLK,USER
12Ø                JP      SEC
121                END
```

```
 1              PROGRAM DECTST(OUTPUT,TAPE6=OUTPUT)

 2        C     TEST CHOLSESKY DECOMPOSITION AND BACK SUBSTITUTIONS ON
 3        C     CDC7600 MINNEAPOLIS

 4              COMMON /TIME/ ICSEC,IRSEC
 5              COMMON/SSTORE/YR(200),YC(200)
 6              COMMON/INFO/N,R(20100),C(20100),BR(200),BC(200),WR(200),WC(200)
 7              COMMON /TIMING/ TDECMP,TBKSB1,TBKSB2,WDECMP,WBKSB1,WBKSB2
 8              WRITE (6,100)
 9        100   FORMAT(1H1,6X,9HDIMENSION,9X,6HDECOMP,5X,10HBACK SUB 1,
               *5X,10HBACK SUB 2,10X,5HTOTAL)
11              N=4
12        1     CONTINUE

13        C     GET NEXT DIMENSION

14              N=NEXDEC(N)
15              IF (N.EQ.0) STOP 777

16        C     FILL UP MATRIX WITH POSITIVE DEFINITE HERMITIAN TEST VALUES

17              LEN=N*(N+1)/2
18              DO 2 J=1,LEN
19              R(J)=1.
20              C(J)=1.
21        2     CONTINUE
22              INDEX=1
23              DUM=10.*N
24              DO 3 J=1,N
25              R(INDEX)=DUM
26              C(INDEX)=0.
27              INDEX=INDEX+J+1
28        3     CONTINUE
29              DO 4 J=1,N
30              BR(J)=1.
31              BC(J)=1.
32        4     CONTINUE

33        C     USE CHOLESKY ROUTINE

34              CALL CHOL

35        C     WRITE (6,102) (WR(J),WC(J),J=1,N)

36        102   FORMAT(8HOWEIGHTS,/,(1X,2E20.10))
37              TMAX=TDECMP+TBKSB1+TBKSB2
38              WMAX=WDECMP+WBKSB1+WBKSB2
39              WRITE(6,101) N,TDECMP,TBKSB1,TBKSB2,TMAX,WDECMP,WBKSB1,WBKSB2,WMAX
40        101   FORMAT(1H0,I15,4F15.3,5X,13HCPU(MILLISEC),/,16X,4F15.3,5X,
               *14HWALL(MILLISEC))
42              GOTO 1
43              END
```

```
      SUBROUTINE CHOL
      COMMON /TIMING/ TDECMP,TBKSB1,TBKSB2,WDECMP,WBKSB1,WBKSB2
      COMMON /TIME/ ICSEC,IRSEC
      COMMON/INFO/N,R(20100),C(20100),BR(200),BC(200),WR(200),WC(200)
      COMMON/SSTORE/YR(200),YC(200)
      DIMENSION UR(20100),UC(20100)
      DIMENSION RECIP(200)
      EQUIVALENCE (UR(1),R(1)),(UC(1),C(1))
C *************************************************************************
C
C THIS ROUTINE SOLVES A SYSTEM OF SIMULTANEOUS EQUATIONS OVER THE
C COMPLEX NUMBERS. THE COEFFICIENT MATRIX MUST BE HERMITIAN AND
C POSITIVE DEFINITE. THE CHOLESKY FACTORIZATION METHOD IS USED, AND THEN
C BACK SUBSTITUTION (TWICE).
C
C VARIABLES . . .
C   N  DIMENSION OF PROBLEM, I.E. MATRIX MUST BE N BY N.
C   R  REAL PART OF INPUT MATRIX. ONLY UPPER TRIANGULAR PART OF
C      MATRIX IS STORED (COLUMNWISE), SO DIMENSION = N*(N+1)/2.
C   C  IMAGINARY PART OF INPUT MATRIX, ALSO STORED COLUMNWISE.
C      DIMENSION = N*(N+1)/2
C   BR REAL PART OF CONSTANT VECTOR, I.E. REAL PART OF B IN THE
C      EQUATION AW=B. DIMENSION = N.
C   BC IMAGINARY PART OF CONSTANT VECTOR, ALSO OF DIMENSION N.
C   WR REAL PART OF SOLUTION VECTOR, I.E. REAL PART OF W IN THE
C      EQUATION AW=B. DIMENSION = N.
C   WC IMAGINARY PART OF SOLUTION VECTOR, ALSO OF DIMENSION N.
C   UR REAL PART OF FACTORED MATRIX. DIMENSION = N*(N+1)/2
C   UC IMAGINARY PART OF FACTORED MATRIX. DIMENSION = N*(N+1)/2
C   YR REAL PART OF RESULTS OF FIRST BACK SUBSTITUTION. DIMENSION = N.
C   YC IMAGINARY PART OF FIRST BACK SUBSTITUTION RESULTS. DIMENSUION = N
C
C *************************************************************************

      T(J,K,ISUB,LOP)=FLOAT(J-K-ISUB)*27.5E-6/FLOAT(LOP)
      LOP=1
      CALL SEC
      IW1=IRSEC
      IC1=ICSEC
      DO 991 JLOP=1,LOP

C
C NOTE. . .
C MATRIX IS ASSUMED UPPER TRIANGULAR
C A(I,J), IN VECTOR STORAGE, IS A(J*(J-1)/2 + I)
C
C COMPUTE FIRST DIAGONAL ELEMENT
C

      UR(1) = SQRT(R(1))
      UC(1) = 0.0
      RECIP(1)=1./UR(1)
      INDEX = 1

C
```

```
      C COMPUTE REST OF FIRST ROW
      C
            NM1 = N - 1
            DO 1Ø J=1,NM1
              INDEX = INDEX + J
              UR(INDEX) = R(INDEX)*RECIP(1)
              UC(INDEX) = C(INDEX)*RECIP(1)
      1Ø    CONTINUE
      C
      C COMPUTE DIAGONAL ELEMENT
      C
            INDEXI = 1
            DO 4Ø I=2,NM1
              IIM1 = INDEXI
              ILOW = IIM1 + 1
              INDEXI = INDEXI + I
              IUP = INDEXI - 1
              IM1 = I - 1
              SUMDIA = Ø.Ø
              DO 2Ø K=ILOW,IUP
                R1 = UR(K)
                C1 = UC(K)
                SUMDIA = SUMDIA + R1*R1 + C1*C1
      2Ø        CONTINUE
              UR(INDEXI) = SQRT(R(INDEXI) - SUMDIA)
              UC(INDEXI) = Ø.Ø
              URDIV = 1.Ø/UR(INDEXI)
              RECIP(I)=URDIV
      C
      C COMPUTE REST OF ROW
      C
              JJM1 = IIM1
              DO 3Ø J=I,NM1
                JJM1 = JJM1 + J
                SUMR = Ø.Ø
                SUMC = Ø.Ø
                DO 25 K=1,IM1
                  R1 = UR(IIM1+K)
                  C1 = UC(IIM1+K)
                  R2 = UR(JJM1+K)
                  C2 = UC(JJM1+K)
                  SUMR = SUMR + R1*R2 + C1*C2
                  SUMC = SUMC + R1*C2 - C1*R2
      25          CONTINUE
                  IJ = JJM1 + I
                  UR(IJ) = URDIV*(R(IJ) - SUMR)
                  UC(IJ) = URDIV*(C(IJ) - SUMC)
      3Ø        CONTINUE
      4Ø    CONTINUE
      C
```

```
      C DO LAST ELEMENT
      C
            ILOW = INDEXI + 1
            IUP = INDEXI + NM1
            SUMDIA = Ø.Ø
            DO 5Ø K=ILOW,IUP
              R1 = UR(K)
              C1 = UC(K)
              SUMDIA = SUMDIA + R1*R1 + C1*C1
      5Ø    CONTINUE
            INDEXI = INDEXI + N
            UR(INDEXI) = SQRT(R(INDEXI) - SUMDIA)
            UC(INDEXI) = Ø.Ø
            RECIP(N)=1./UR(INDEXI)
      991   CONTINUE
            CALL SEC
            IW2=IRSEC
            IC2=ICSEC
            DO 992 JLOP=1,LOP
      C
      C FIRST BACK SUBSTITUTION, USING CONJUGATE TRANSPOSE
      C
            DIV=RECIP(1)
            YR(1) = BR(1)*DIV
            YC(1) = BC(1)*DIV
            INDEX = 1
            ILOW = Ø
            IHIGH = Ø
            DO 7Ø I=2,N
              ILOW = IHIGH + 2
              IHIGH = IHIGH + I
              SUMR = Ø.Ø
              SUMC = Ø.Ø
              K = Ø
              DO 6Ø J=ILOW,IHIGH
                K = K + 1
                SUMR = SUMR + YR(K)*UR(J) + YC(K)*UC(J)
                SUMC = SUMC - YR(K)*UC(J) + YC(K)*UR(J)
      6Ø      CONTINUE
              INDEX = INDEX + I
              DIV=RECIP(I)
              YR(I) = (BR(I) - SUMR)*DIV
              YC(I) = (BC(I) - SUMC)*DIV
      7Ø    CONTINUE
      992   CONTINUE
            CALL SEC
            IW3=IRSEC
            IC3=ICSEC
            DO 993 JLOP=1,LOP
      C
      C SECOND BACK SUBSTITUTION
      C
```

```
197             M = N
198             I = N*(N+1)/2
199       80    DIV=RECIP(M)
200             WR(M) = YR(M)*DIV
201             WC(M) = YC(M)*DIV
202             TR = WR(M)
203             TC = WC(M)
204             M = M - 1
205             DO 90 J=1,M
206               K = M - J
207               L = I - J
208               YR(K+1) = YR(K+1) - UR(L)*TR + UC(L)*TC
209               YC(K+1) = YC(K+1) - UR(L)*TC - UC(L)*TR
210       90    CONTINUE
211             I = I - M - 1
212             IF(I.NE.1)GOTO 80
213             DIV=RECIP(1)
214             WR(1) = YR(1)*DIV
215             WC(1) = YC(1)*DIV
216       993   CONTINUE
217             CALL SEC
218             IW4=IRSEC
219             IC4=ICSEC
220             DO 994 JLOP=1,LOP
221       994   CONTINUE
222             CALL SEC
223             IW=IRSEC-IW4
224             IC=ICSEC-IC4
225             TDECMP=T(IC2,IC1,IC,LOP)
226             WDECMP=T(IW2,IW1,IW,LOP)
227             TBKSB1=T(IC3,IC2,IC,LOP)
228             WBKSB1=T(IW3,IW2,IW,LOP)
229             TBKSB2=T(IC4,IC3,IC,LOP)
230             WBKSB2=T(IW4,IW3,IW,LOP)
231             RETURN
232             END
```

```
233             FUNCTION NEXDEC(N)
234             IF (N.GE.2Ø) GOTO 1
235             NEXDEC=2Ø
236             RETURN
237       1     IF (N.GE.2ØØ) GOTO 2
238             NEXDEC=N+2Ø
239             RETURN
24Ø       2     NEXDEC=Ø
241             RETURN
242             END
```

```
243                 IDENT   SEKOND
244                 USE     /TIME/
245       ICSEC     BSS     1
246       IRSEC     BSS     1
247                 USE     *
248                 ENTRY   SEC
249       SEC       BSS     1
25Ø                 RTIME   IRSEC,7CLK
251                 TIME    ICSEC,7CLK,USER
252                 JP      SEC
253                 END
```

Appendix H. A Necessary Condition for the Nonsingularity of a Sample Covariance Matrix

Given a collection $\{\underline{S}^{(j)}\}_{j=1}^{m}$ of m complex n-dimensional sample vectors, their sample covariance matrix is defined by

$$\underline{M} = \sum_{k=1}^{m} \underline{M}^{(k)} = \sum_{k=1}^{m} \underline{S}^{(k)^*} \underline{S}^{(k)} \tag{1}$$

where $\underline{S}^{(k)^*}$ is the conjugate transpose of $\underline{S}^{(k)}$, so that the ij^{th} component of $\underline{M}^{(k)}$ is

$$M_{ij}^{(k)} = \overline{S}_i^{(k)} S_j^{(k)} . \tag{2}$$

We claim that M is singular if $m < n$, i.e., if the number of sample vectors is less than their dimension.

Proof: First we need a definition and a lemma.

Definition: The rank of a matrix $\underline{A}$ is the dimension of the space spanned by its columns.

Note that a square matrix $\underline{A}$ is nonsingular if and only if its dimension equals its rank, rank($\underline{A}$). This is because $\underline{A}$ is nonsingular if and only if its columns (or rows) are independent, i.e., ℓ columns span a space of dimension ℓ for any subset of ℓ columns.

Lemma: Rank $(\underline{A} + \underline{B}) \leq$ rank $(\underline{A})$ + rank (B).

Proof: Given a set of vectors which span a space of dimension n_a and a set $\mathcal{B}$ of vectors which span a space of dimension n_b, the set of vectors $\mathcal{A} \cup \mathcal{B}$ clearly spans a space of dimension $\leq n_a + n_b$, so any set S of linear combinations of vectors in $\mathcal{A} \cup \mathcal{B}$ spans a space of dimension $\leq n_a + n_b$. Now let $\mathcal{A}$ = {columns of $\underline{A}$}, $\mathcal{B}$ = {columns of $\underline{B}$}, and S = {columns of $\underline{A} + \underline{B}$} and the result follows. Q.E.D.

Corollary: Rank $\left(\sum_{k=1}^{m} \underline{M}^{(k)} \right) \leq \sum_{k=1}^{m}$ rank $[\underline{M}^{(k)}]$.

Proof: Follows immediately from the lemma by induction. Q.E.D.

Now we may continue with the main proof.

Since the columns of $\underline{M}^{(k)}$ are all multiples of the vector $\underline{S}^{(k)^*}$ by Eq. (2), the vector $S^{(k)^*}$ spans the space spanned by the columns of $\underline{M}^{(k)}$ so that space has dimension = rank $(\underline{M}^{(k)} \leq 1)$(the rank could be 0 if all the components of $S^{(k)}$ are 0). Since

$$\underline{M} = \sum_{k=1}^{m} \underline{M}^{(k)}$$

NOTE: This proof presents only a necessary condition for the nonsingularity of the sample covariance matrix, not a sufficient one. In other words, the matrix may still be singular even if the number of samples exceeds the dimension. The interested reader is referred to "Rapid Convergence Rate in Adaptive Arrays" by Reed, Mallett and Brennan*, for a sufficient condition on the number of samples for good results.

*Reed, I. S., J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-10, No. 6, 1974, pp. 853-863.

Appendix I. Number of Bits Needed for Sample Covariance Matrix Calculation

This appendix deals with the lower bound on computational complexity of forming the sample covariance matrix and the accuracy to which it can be computed. It also includes a section on how these bounds are related and may be changed due to machine limitations.

The i^{th} sample covariance matrix can be defined as

$$M^i = X^{i*}X^i + M^{i-1} \qquad i = 1,2,...$$

where

X^i is the i^{th} input voltage sample vector (i.e., row vector)

X^{i*} is the conjugate transpose of X^i

M^i is the covariance matrix

M^o elements all have value of zero.

We will also define the following:

N is the number of weights in the system; this is equal to the number of elements in X.

S is the number of samples used to form M. It has been shown that by setting S = 2*N, the weights will be computed within 3 dB of optimal.*

A. COMPUTATIONAL COMPLEXITY

Since M is Hermitian, only the lower or upper triangular region needs to be computed. We will assume that the upper triangular region is being computed.

*Reed, I. S., J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-10, No. 6, 1974, pp. 853-863.

This section of the matrix contains N*(N+1)/2 elements. Each element of the vector X is only multiplied by a conjugate of an element of X to update a location in M. A program for a serial machine would look like the following:

```
DO I = 1,N
   DO J = 1,N
      M(I,J) = CONJUGATE (X(I))*X(J) + M(I,J)
   END DO
END DO
```

In actual practice, only N(N+1)/2 multiplies must be computed, with N(N+1)/2 adds needed to update M. A program for a serial machine would now look like the following, assuming that the elements of M are stored column-wise:

```
INX = 0
DO I = 1,N
   INX = INX + I
   JNX = INX
   DO J = I,N
      M(JNX) = CONJUGATE (X(I))*X(J) + M(JNX)
      JNX = JNX + J
   END DO
END DO
```

B. COMPUTATIONAL ACCURACY

Accuracy will be expressed in number of bits. To compute the number of bits needed, two quantities must be known: 1) the sample size, S, and 2) the number of bits from the analog to digital converter (ADC), B.

Whenever a binary add is performed on two words of a and b bits long, the resultant sum can require MAX (a,b)+1 bits. Whenever a binary multiply is performed on two words of a and b bits long, the resultant product can require a+b bits.

Elements of X are expressed as a real and imaginary part, each B bits long. Performing multiplications will require 2 B bits. Each complex multiply provides an add, so each individual term of $X^{*}X$ will require (2*B + 1) bits.

These values must be added to the previous computed values S-1 times. In vector notation the formula is

$$M_{ij} = \sum_{k=1}^{S} X_i^{*k} X_j^k \quad .$$

Therefore the total number of bits required is $\lceil LOG_2\ S*(2^{2*B+1}-1)+1 \rceil$ where $\lceil X \rceil$ means the smallest integer greater than or equal to X (the ceiling function). This formula is derived as follows: The maximum number which can be expressed in k bits is 2^k-1. If we are to sum values of b bits, the worst case is 2^b-1.

Let

k be no. of bits required for answer

b be no. of bits in items to be summed

S be no. of items to be summed

k,b, & s all integers, c,b,s>0

$S*(2^b-1) \leq 2^k-1$.

Solve for k, as follows:

$S*(2^b-1) + 1 \leq 2^k$

$LOG_2(S*(2^b-1) + 1) \leq k$

$\lceil LOG_2(S*(2^b-1) + 1) \rceil = k$ because all variables are integers.

Solve for S

$$S \leq \frac{2^k-1}{2^b-1}$$

$$S = \left\lfloor \frac{2^k-1}{2^b-1} \right\rfloor = \sum_{i=1}^{\left\lfloor \frac{k}{b} \right\rfloor} 2^{k-ib} .$$

To choose some numbers for examining this function let B = 12 bits and S = 2*N, when N = 200.

$$\lceil LOG_2\ S*(2^{2*B+1}-1)+1 \rceil$$

$$\lceil LOG_2\ 400*(2^{25}-1)+1 \rceil$$

34 bits required.

Table I-1 lists different computers and their mantissa length in bits. The table also lists the maximum N allowed with S = 2*N.

C. WHAT TO DO IF MACHINE'S ACCURACY IS INSUFFICIENT

There are a couple of approaches to this problem. Before any program is discussed, however, a deeper understanding of the calculations in Section B is required.

In Section B, we calculated the number of bits needed to ensure that no errors are introduced into the calculations; all values were treated as fixed point. Almost all advanced computers have floating-point hardware on them and, in fact, this hardware will be used during the inversion process to minimize errors introduced by division.

METHOD 1

One solution to the accuracy problem is to ignore it and let the floating-point hardware take care of it. There is one problem with this approach, which only occurs when the number of samples is very large. All multiplies are on similar-magnitude numbers (namely B bits) and all adds to perform complex multiplies are on similar-magnitude numbers (namely 2*B bits). After we have added up a large number of samples, the magnitude of the values for M_{ij} may be much larger than (2*B+1) bits. In that case, the addition may only add in part of this product, or worse, none of it.

As an example, assume that the mantissa is 30 bits long with exponent, and B = 12 bits as before.

How many samples can be added before we lose 5 bits of a product term? That is the same as asking, How many samples can we accumulate in 35 bits? By using the formula from Section B we have

$$S = \left\lfloor \frac{2^{k}-1}{2^{2*B+1}} \right\rfloor$$

$$S = \left\lfloor \frac{2^{35}-1}{2^{25}-1} \right\rfloor$$

$$S = 2^{10} = 1024 \text{ samples} .$$

Another question is, How many samples can be summed before an error is introduced ? Again with 30-bit mantissa, we apply the same formula:

$$S = \left\lfloor \frac{2^{k}-1}{2^{2*B+1}-1} \right\rfloor$$

$$S = \left\lfloor \frac{2^{30}-1}{2^{25}-1} \right\rfloor$$

$$S = 2^{5} = 32 \text{ samples} \quad .$$

We can also compute when any additional samples will be simply ignored, i.e., will not change the running sum at all. Again with the same assumptions as before

Total no. of bits required = 30 + 25 = 55

$$S = \left\lfloor \frac{2^{55}-1}{2^{25}-1} \right\rfloor \quad ,$$

$$S = 2^{30} + 2^{5} \text{ samples} \ ,$$

which is indeed a large number, but by the time we have reached this limit, the cumulative error is very large.

The inversion process introduces other errors, though some errors in the covariance matrix may be acceptable due to the accuracy requirements of the weights. Whether the existing floating-point hardware is sufficient or not will depend upon the particular system specifications.

METHOD 2

There is another technique to limit the size of the numbers. The sample covariance matrix represents the expected value between the i^{th} and j^{th} component of the signal.

The expected value is approximated by averaging a set of samples together. This can be represented as

$$M_{ij} = \frac{1}{S} \sum_{k=1}^{S} X_i^{k*} X_j^k.$$

The division has the effect of reducing the number of bits required. Instead of computing the entire sum, partial sums can be computed and averaged. The computation is now

$$M_{ij} = \sum_{m=0}^{\frac{S}{s}-1} \left(\frac{1}{s} \sum_{k=ms+1}^{ms+s} X_i^{*k} X_j^k \right).$$

By choosing s appropriately, the total number of bits required can be reduced. Again, choosing s is based upon the particular system (ADC, computer, accuracy of weights, etc.). By doing this calculation we add $\left[\frac{S}{2} * \frac{(N+1)}{2}\right]$ divisions to compute M.

It should be noted that if M is produced by the method of Section B, it may be advisable to divide each element by S to reduce the number of bits required. This is allowable because it only changes all the weights by a constant factor, and due to the weighting processor, multiplicative constant factors have no effect upon choosing optimal weights, i.e., for given optimal weights, W, all constant multiples are also optimal.

It should also be noted that for all divisions, the divisors are real numbers.

Let us run through an example using averaging. Assume mantissa length is 30 bits, B = 12. What is the maximum S and s that can be used?

$$s = \left\lfloor \frac{2^{30}-1}{2^{2*B+1}-1} \right\rfloor$$

$$s = \left\lfloor \frac{2^{30}-1}{2^{25}-1} \right\rfloor \quad .$$

$$s = 2^5 = 32$$

Each division by s reduces the product back to 30 - 5 = 25 bits. We now must compute how many sums of 25-bit numbers are needed to reach the 30-bit limit.

$$\text{No. of sums} = \left\lfloor \frac{2^{30}-1}{2^{25}-1} \right\rfloor \quad .$$

$$\text{No. of sums} = 32$$

$$S = \text{No. of sums} * s$$

$$32 * 32 = 1024 \quad .$$

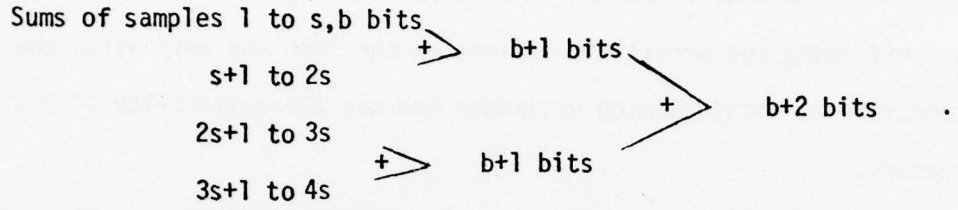
If we assume S = 2*N, then this method with the number of bits specified will handle a system of 512 weights.

METHOD 3

Methods 1 and 2 can of course be combined. The main problem with Method 1 is the errors introduced by adding numbers of greatly differing magnitudes. This can be overcome by calculating M in the following manner.

$$M_{ij} = \sum_{m=0}^{\frac{S}{s}-1} \sum_{k=ms+1}^{ms+s} X_i^{k*} X_j^{k} \quad .$$

By calculating sums of s terms and adding them together, we will be adding numbers of similar magnitude. In the extreme, the additions will be performed in a tree-like manner as shown in the following example.



Assuming only b bits in the mantissa, this procedure minimizes errors.

At a minimum this method, as well as Method 2, requires N(N+1)/2 more storage locations.

METHOD 4

In this method, the entire covariance matrix calculation is done in fixed-point math. On the machines we are concerned with, fixed-point is implemented as integer math.

On machines such as the PEPE and STARAN which have large bit length integers (PEPE can use 46 bits and STARAN has variable length), this method seems better than losing accuracy by using a short floating-point format.

The STARAN is unique among the computers being studied in that there is no fixed word length for numbers. A STARAN word can be divided into fields of any number of bits. The STARAN word length is 256 bits on the old model and 9K on the new model. The STARAN performs integer instructions in bit serial manner, so that ADD and MULTIPLY times are directly a function of the number of bits. To properly use this machine, the bounds on bit length will be used to minimize running time.

D. SUMMARY

We notice that the general-purpose scientific machines all have sufficient bits to handle the covariance matrix calculation. We will take advantage of this by performing the detail simulations on the 7600 and only using the supercomputers to obtain timing estimates and see the suitability of their architecture.

The two exceptions to the above are the STARAN and PEPE. In these two cases the calculations will be done in integer and then converted to floating-point for the inversion process. These integer calculations can be simulated on the 7600, so again these machines only need to be examined for speed considerations and the applicability of their hardware architecture to the problem.

One item of research is to compute what the loss is on the 24- and 23-bit mantissa machine. The reasons for using the smaller mantissa length is speed, but architecture considerations will not change.

Table I-1. Attainable Accuracy for Covariance Matrix (ADC is 12 Bits, Sample Size = 2 x No. of Weights).

COMPUTER	NO. OF BITS IN FLOATING-POINT MANTISSA	MAXIMUM NO. OF WEIGHTS
CDC 7600		
SINGLE PRECISION	48	2^{22}
DOUBLE PRECISION	96	$2^{70} + 2^{46} + 2^{21}$
CDC STAR-100		
HALF PRECISION	23	TOO SMALL -0
SINGLE PRECISION	47	2^{21}
CRAY-I		
SINGLE PRECISION	48	2^{22}
TI ASC		
SINGLE PRECISION	24	TOO SMALL -0
DOUBLE PRECISION	88	$2^{62} + 2^{37} + 2^{12}$
ILLIAC IV		
SINGLE PRECISION	24	TOO SMALL -0
DOUBLE PRECISION	48	2^{22}
PEPE		
SINGLE PRECISION	23	TOO SMALL -0
STARAN	VARIABLE PRECISION INTEGER	N/A

Appendix J. Parallel Implementations of Computing the Sample Covariance Matrix

We have already outlined the computations needed to compute the sample covariance matrix. We will now discuss methods for computing it. We will first show a parallel special-purpose hardware approach. We will use this as a model to develop approaches for parallel and vector machines.

Let

X^i be the i^{th} voltage vector

X_{iI} be the I part of the i^{th} element of X

X_{iQ} be the Q part of the i^{th} element of X

M^i be the i^{th} sample covariance matrix.

Then

$M^i = X^{i*} X^i + M^{i-1}$ where M^0 contains all zero elements.

This can be rewritten as

$$M^i_{ijI} = X^i_{iI} X^i_{jI} - X^i_{iQ} X^i_{jQ} + M^{i-1}_{ijI}$$

$$M^i_{ijQ} = X^i_{iQ} X^i_{jI} + X^i_{iI} X^i_{jQ} + M^{i-1}_{ijQ} \quad .$$

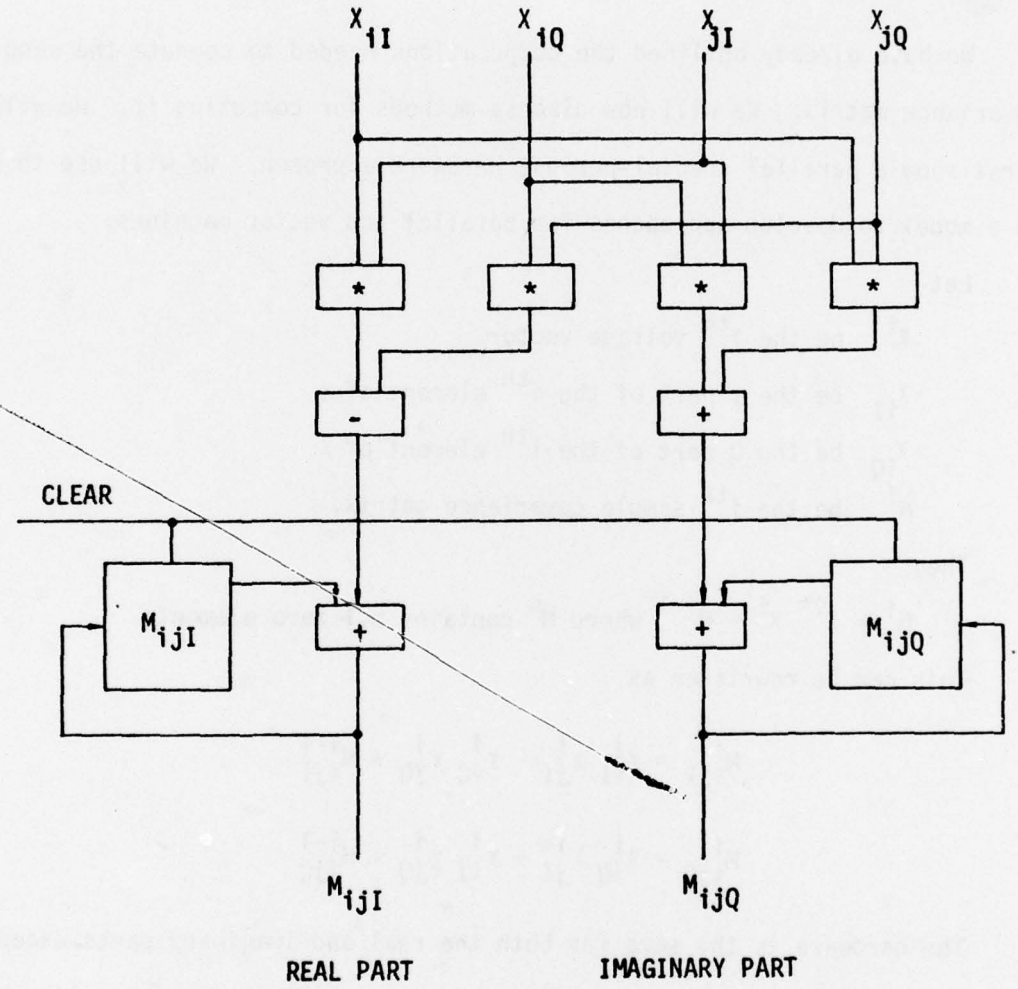
The hardware is the same for both the real and imaginary parts except for an adder/subtractor. The hardware is also identical for every ij pair. This commonality will be exploited in terms of software designs and could be exploited in hardware designs by having common replacement modules.

AD-A054 358 TECHNOLOGY SERVICE CORP SANTA MONICA CALIF F/G 17/9
MULTIDOMAIN ALGORITHM EVALUATION. VOLUME II.(U)
APR 78 W C LILES, J C DEMMEL, I S REED F30602-76-C-0319
UNCLASSIFIED TSC-PD-B525-1-VOL-2 RADC-TR-78-59-VOL-2 NL

3 OF 3
AD
A054358

END
DATE
FILMED
6 -78
DDC

The following circuitry will perform these operations:

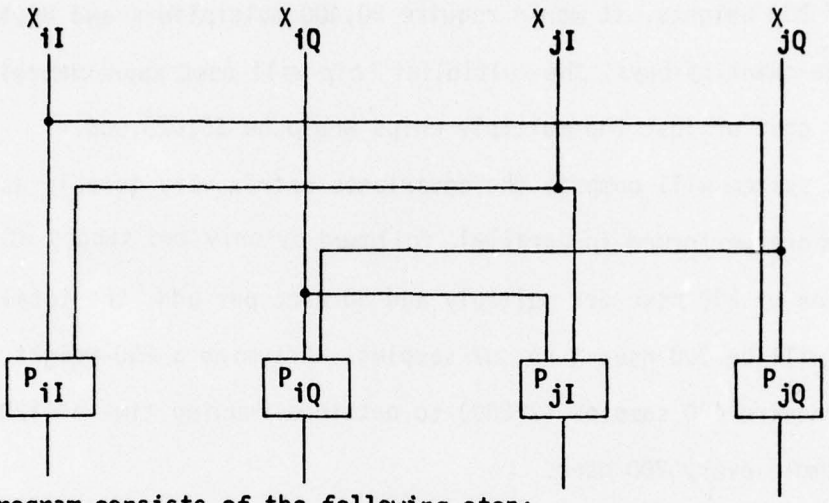


This system is not practical for a large-scale problem: e.g., if we have a system of 200 weights, it would require 80,400 multipliers and 80,400 adders. With large-quantity buys, the multiplier chip will cost approximately $70. The total cost of just the multiply chips would be $5,628,000.

This system will compute the covariance matrix very quickly as all multiplies are performed in parallel, followed by only two stages of adding. Using times of 200 nsec per multiply and 50 nsec per add, the total time required will be 300 nsec * no. of samples. Assuming a 200-weight system, we will require 400 samples (2*200) to obtain a running time of 120 μsec with a sample every 200 nsec.

Since the limiting factor in terms of sampling rate is the multiplier rates, the system can be modified by changing multipliers so that there are sufficient multipliers for each ij pair to deliver results to the adders at their rate. This can be illustrated better in a timing diagram. We will use 4 multipliers of 200 nsec and one adder of 25 nsec. This system will sample at a rate of one sample every 50 nsec. It should be noted that this system requires 4 times as many multipliers.

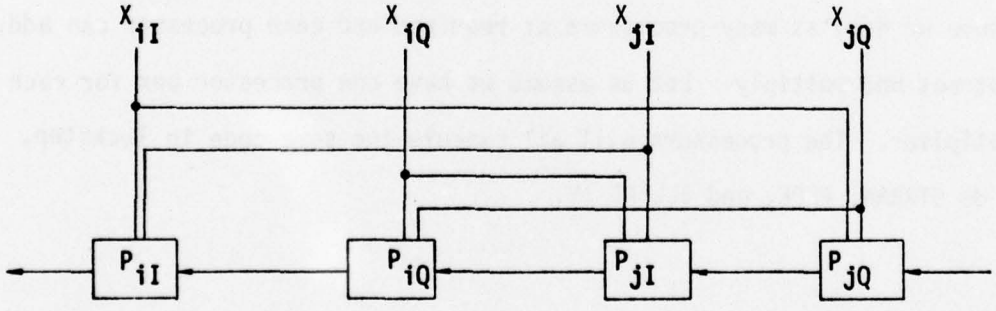
To put the sample algorithm on a parallel processor, we will first assume we have as many processors as required and each processor can add, subtract and multiply. Let us assume we have one processor box for each multiplier. The processors will all execute the same code in lockstep, as do STARAN, PEPE, and ILLIAC IV.



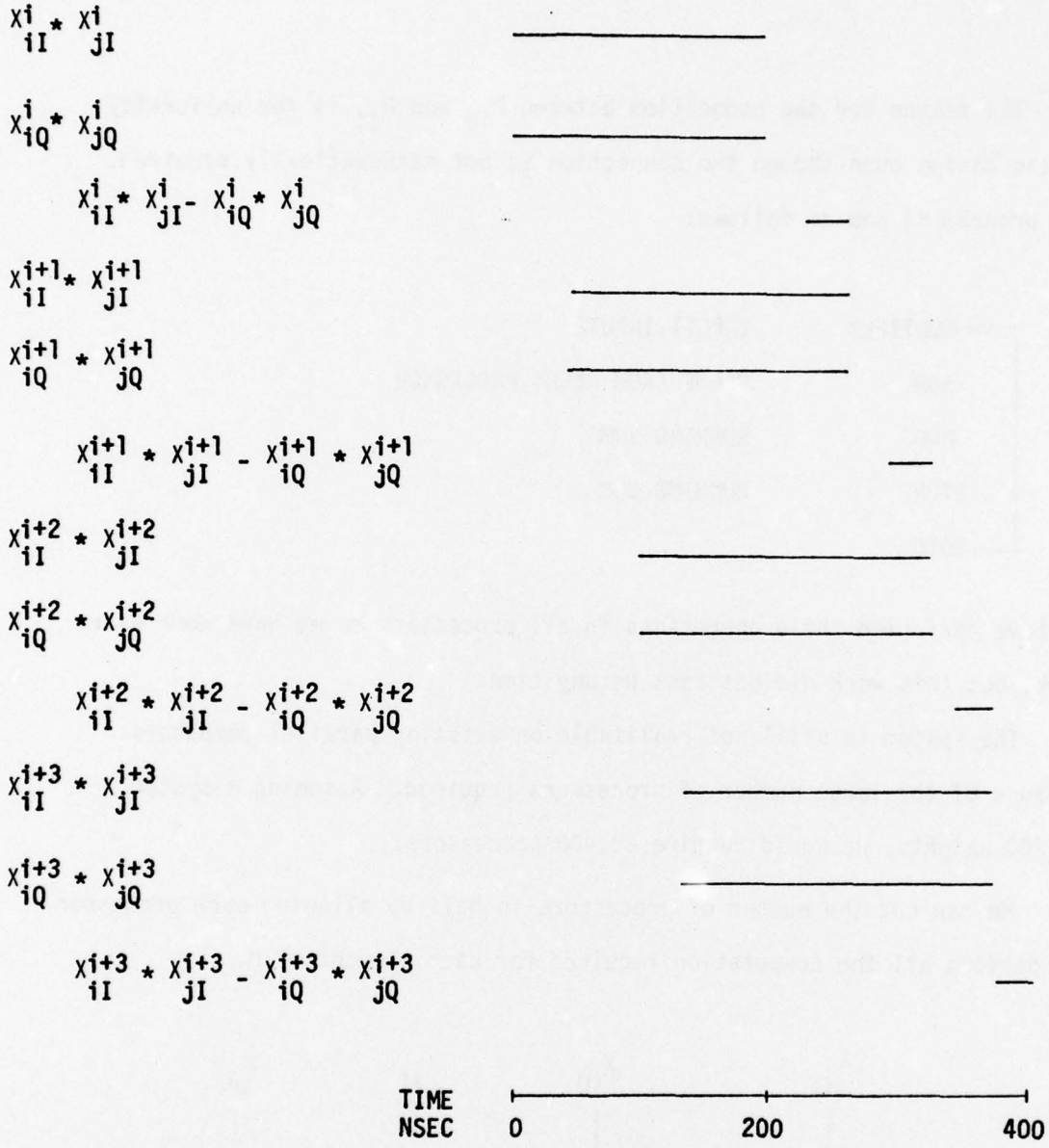
If the program consists of the following step:

MULTIPLY INPUT1,INPUT2

We have completed the multiply but not the add as the partial results are in different processors. To overcome this problem, we will let processors communicate with their neighbor. This is allowable in the STARAN and ILLIAC IV but not the PEPE.



CALCULATION



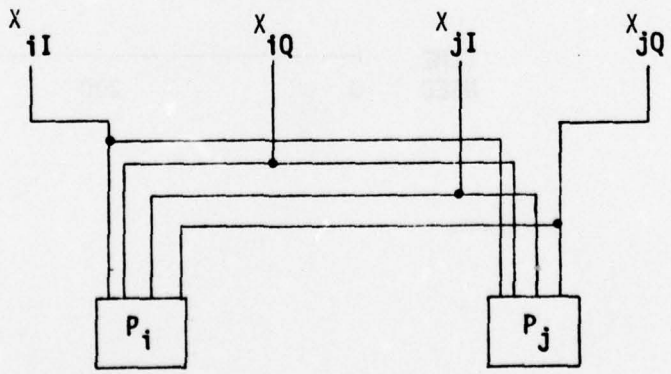
The reason for the connection between P_{iQ} and P_{jI} is for uniformity in the design even though the connection is not mathematically required. The program is now as follows:

```
→MULTIPLY      INPUT1,INPUT2
 ADD           VALUE FROM RIGHT PROCESSOR
 ADD           RUNNING SUM
STORE          RUNNING SUM
 GOTO
```

We have performed these operations in all processors so we have done extra work, but this work did not cost us any time.

The system is still not realizable on existing parallel computers because of the large number of processors required. Assuming a system of 200 weights, we would require 80,400 processors.

We can cut the number of processors in half by allowing each processor to perform all the computation required for each element of M.



The program will be as follows:

```
┌►MULTIPLY        INPUT1,INPUT2
│ STORE           TEMP1
│ MULTIPLY        INPUT3,INPUT4
│ ADD             TEMP1
│ ADD             RUNNING SUM
│ STORE           RUNNING SUM
└─ GOTO
```

This procedure does not require any processor-to-processor communication, so it is suitable for the PEPE.

Unfortunately, the number of processors required is still too large for existing machines for the problem we are interested in. The obvious way of fitting this problem to existing machines is to simply partition the program into subproblems which can be handled. In the case of 200 weights, we have 20,100 complex terms to compute. If we have N processors available, we will compute N complex terms at a time and iterate 20100/N times.

Another approach known as the "global method," which is based upon the STARAN architecture, is as follows:

Set i to 1

Load X_{iI} into the global register

Multiply all appropriate elements by this value

Load X_{iQ} into the global register

Multiply all appropriate elements by this value

Do parallel adds

i ← i + 1

If ($i \leq N$)

The execution time of this algorithm is a function of N. If the number of processors, M, is greater than N, then the execution time is proportional to N. If N is greater than M, which will usually be the case for large M, we will do M elements at a time. The execution time is then proportional to the number of groups which the problem must be devoted to.

For a triangular system,

$$G = (P+1)\left(\frac{MP}{2} + N - P*M\right) \text{ where}$$

$$P = \left\lfloor \frac{N}{M} \right\rfloor .$$

For the complete matrix,

$$G = N\left\lceil \frac{N}{M} \right\rceil .$$

Both of these equations reduce to N when $M \geq N$. This means that in this case, the entire matrix can be obtained in the same time as only the triangular matrix.

The drawback to this approach is that not all parallel processors will be active at all times, but this multiply is faster than the multiply which uses two values in the same word. We would like a system which always uses the maximum number of available processors so that execution time will be minimal.

This leads to another method, to be known as the "packed method." We will illustrate these approaches by means of examples.

Completely Parallel Computation Approach

Assume the number of processors is greater than the number of weights.

N = 3 no. of weights

M = 4 no. of processors

Processor	A	B	C	D								
1	X_{1I}	X_{1Q}	X_{1I}	X_{1Q}	M_{11I}	M_{11J}	X_{2I}	X_{2Q}	X_{3I}	X_{3Q}	M_{23I}	M_{23J}
2	X_{1I}	X_{1Q}	X_{2I}	X_{2Q}	M_{12J}	M_{12J}	X_{3I}	X_{3Q}	X_{3I}	X_{3Q}	M_{33I}	M_{33J}
3	X_{1I}	X_{1Q}	X_{3I}	X_{3Q}	M_{13I}	M_{13J}						
4	X_{2I}	X_{2Q}	X_{2I}	X_{2Q}	M_{22I}	M_{22J}						

Computation Group 1

Time Steps 1-8

A*C - B*D

A*D + B*C

Computation Group 2

Time Steps 9-16

The upper triangular matrix is updated in 16 time steps. We notice that in the second group, two processors are available for work. We will now rework the example, using 3 processors.

Processor												
1	X_{1I}	X_{1Q}	X_{1I}	1Q	M_{11I}	M_{11J}	2I	2Q	2I	2Q	M_{22I}	M_{22J}
2	1I	1Q	2I	2Q	M_{12I}	M_{12J}	2I	2Q	3I	3Q	M_{23I}	M_{23J}
3	1I	1Q	3I	3Q	M_{13I}	M_{13J}	3I	3Q	3I	3Q	M_{33I}	M_{33J}
	Group 1						Group 2					
	Time Steps 1-8						Time Steps 9-16					

An interesting observation can be made: namely, that the execution time is the same with one less processor. It can also be shown that the evaluation time will also be the same with 5 processors.

The total number of complex terms is N(N+1)/2 for the triangle form of the matrix and N^2 for the full matrix. If we have M processors, then the number of groups, G, will be

$$G = \left\lceil \frac{N(N+1)}{2M} \right\rceil \quad \text{for triangular matrix}$$

$$G = \left\lceil \frac{N^2}{M} \right\rceil \quad \text{for the complete matrix}$$

where $\lceil x \rceil$ is smallest integer greater than or equal to x, i.e., the ceiling function.

These equations can be used in the following ways:

Given the maximum time, T, allowed to compute the covariance matrix and the dimension, N, what is the minimum number of processors required?

We will now relate number of processors, M, and number of weights, N, to determine number of groups, G, needed to perform the covariance update. The running time is a constant, C, times the number of groups.

The time per group is t, then the number of groups required is $G = \lceil T/t \rceil$. We then solve

$$G = \left\lceil \frac{N(N+1)}{2M} \right\rceil \quad \text{given G \& N}$$

$$M = \left\lceil \frac{N(N+1)}{2G} \right\rceil \quad .$$

Because of the ceiling function the total time may be less than T.

If a machine such as STARAN, PEPE or ILLIAC IV is available, then M will be chosen as the maximum available.

If a machine is to be designed then the above computation will be performed.

Because for a given G and N, more than one M may satisfy the equations, questions of designing for reliability come up.
For example, let N = 200, G = 201

Then M can range from 76 to 100. If the system is constructed with 76 processors and one breaks, then G will increase to 268 with a corresponding increase in running time. If, however, 100 processors were constructed, then up to 24 processors could break without degrading system performance. Neither extreme is probably practical. By calculating the MTBF, MTTR and MAXTTR of processors, one can determine the number of processors above the minimum number of processors which should be specified.

Another question of reliability is, What occurs if N becomes smaller, such as if a receiver breaks? By looking at the equation $G = \left\lceil \frac{N(N+1)}{2M} \right\rceil$, it is obvious that for N_1 and N_2 such that $N_1 < N_2$ and

$$G_1 = \left\lceil \frac{N_1(N_1+1)}{2M} \right\rceil \text{ and } G_2 = \left\lceil \frac{N_2(N_2+1)}{2M} \right\rceil .$$

Then $G_1 \leq G_2$. This means that for M fixed, there will be no increase in running time as N decreases.

Comparing the global and packed methods for a triangular matrix, we have the following:

N = 200, M = 256 (1 STARAN array)

Global	Packed
$P = \left\lfloor \frac{N}{M} \right\rfloor$	$G = \left\lceil \frac{N(N+1)}{2M} \right\rceil$
$G = (P+1)\ (MP/2 + (M=P_M))$	= 79
P = 0	
G = 200	

This implies that the packed form is worthy of study. Currently the algorithms are not known for implementing it on the STARAN and PEPE. Also in the example given N<M so we are not utilizing N-M = 56 processors.

Appendix K. Parallel Direct Implementations to Solve $MW = \bar{S}$ with $O(N^2)$ Processors

These implementations were developed by noticing that M has N^2 elements, and that it would be possible to assign one processor to each element. If only the upper or lower triangular matrix is stored, then N(N+1)/2 elements are required. By assigning one element to each processor, we hope to obtain a speed increase over sequential or parallel techniques with O(N) processors.

We will look in detail at two algorithms: Gauss-Jordan and Cholesky. The reason for selecting Gauss-Jordan is its inherent parallelism of selecting a two and then eliminating on all other rows. Cholesky was chosen because it requires only half of the matrix, and we feel the difference between N^2 processors and N(N+1)/2 processors is sufficiently large for large N and that this algorithm is worth exploring. For other implementations see Sameh and Kuck.*

Gauss-Jordon Implementation

We assign to each processor an element of M_1 processor. p contains element M_{ij} such the $p = (i-1)N + j$; $1 \leq i, j \leq N$. This requires N^2 processors. We use another N processor to hold the elements of the steering vector.

We will illustrate the technique on a 3 x 3 matrix, Figure K-1.

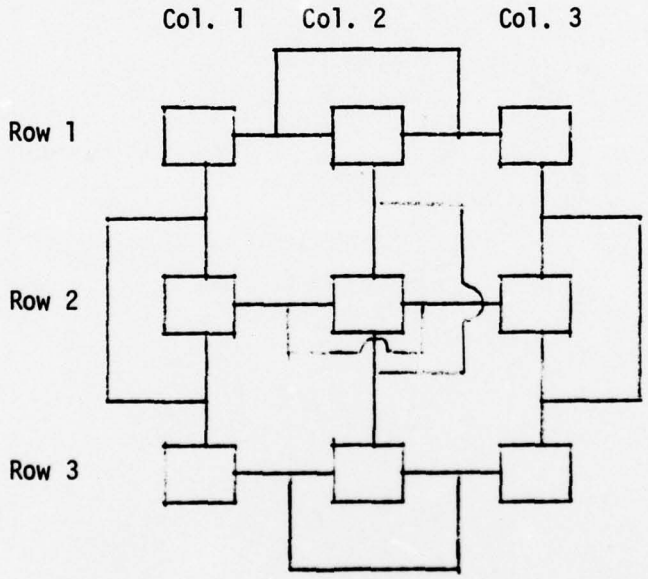
---

*Sameh, A. H., and D. J. Kuck, Linear System Solvers for Parallel Computers, Department of Computer Science, Report No. UIUCDCS-R-75-701, University of Illinois at Urbana-Champaign, February 1975.

Processor	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
Original	a_{11}	a_{12}	a_{13}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}
Temporaries	0	0	0	0	0	0	0	0	0
Normalized	1	a_{12}/a_{11}	a_{13}/a_{11}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}
	0	0	0	0	0	0	0	0	0
Expand (outer product)	1	a'_{12}	a'_{13}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}
	0	0	0	$1(a_{21})$	$a'_{12}(a_{21})$	$a'_{13}(a_{11})$	$1(a_{31})$	$a'_{12}(a_{31})$	$a'_{13}(a_{31})$
Subtract	1	a'_{12}	a'_{13}	0	$a_{21}-a'_{12}a_{21}$	$a_{23}-a'_{13}(a_{21})$	0	$a_{32}-a'_{12}(a_{31})$	0
	0	0	0	0	0	0	0	0	0

Figure K-1 A Parallel Implementation of Gauss-Jordan with $O(N^2)$ Processors When N = 3

We have now eliminated on the first row in three steps. Since there are N rows, Gauss-Jordan will take 3N steps. All of these steps are straightforward except the expand (outer product) step. This operation requires movement of data between processors, if only a global register is available, the data movement would require moving 2N-2 data items and 3N-3 processor enables for each row. Since there are N rows, data movement will be an $O(N^2)$ process and the direct methods using O(N) processors are comparable. What is required is a special interconnect structure which is able to perform the other product in time independent of N, preferably in one or two steps. This interconnect structure is possible and is in fact similar to that of the ILLIAC IV. Again, by using a 3 x 3 matrix, the interconnect strength looks like:



To propagate the values as in the previous example, we move all data vertically in one step and all data horizontally in one step. This can also be accomplished if there is a global register for each row and column.

If the ILLIAC IV structure is used, then N-1 steps would be required for vertical movement and N-1 steps would be required for horizontal movement.

Although this discussion has dealt with performing G-J, as shown earlier, by simply adjoining the steering vector to the matrix M, we will obtain the weights at the same time.

Cholesky Implementation

In this system, we have N(N+1)/2 parallel processors. Again we will demonstrate the technique on a 3 x 3 matrix, Figure K-2.

Processing	p_1	p_2	p_3	p_4	p_5	p_6
Original	a_{11}	a_{21}	a_{31}	a_{22}	a_{32}	a_{33}
SQRT	$\sqrt{a_{11}}$	a_{21}	a_{31}	a_{22}	a_{32}	a_{33}
Divide	$\sqrt{a_{11}}$	$a_{21}/\sqrt{a_{11}}$	$a_{31}/\sqrt{a_{11}}$	a_{22}	a_{32}	a_{33}
Outer Product Subtraction	$\sqrt{a_{11}}$	$a_{21}/\sqrt{a_{11}}$	$a_{31}/\sqrt{a_{11}}$	$a_{22}-\left(\frac{a_{21}}{\sqrt{a_{11}}}\right)\left(\frac{\overline{a_{21}}}{\sqrt{a_{11}}}\right)$	$a_{32}-\left(\frac{a_{31}}{\sqrt{a_{11}}}\right)\left(\frac{\overline{a_{21}}}{\sqrt{a_{11}}}\right)$	$a_{33}-\left(\frac{a_{31}}{\sqrt{a_{11}}}\right)\left(\frac{\overline{a_{31}}}{\sqrt{a_{11}}}\right)$

Figure K-2 A Parallel Implementation of Cholesky with $O(N^2)$ Processors When N = 3

As in the previous section on G-J, the outer product requires an interconnect structure to use this system to advantage.

This process leaves the L decomposition of M in the N(N+1)/2 processors. We must still solve for the weights. This can be done by two back substitutions or adjoining $\bar{S}$ to the original matrix and performing one back substitution as shown in Part 4. A back substitution can be performed in N processors as outlined in the discussion on PEPE. We can use either N processors or the original N(N+1)/2 processors. If another set of processors is used, however, we can pipeline the processors to work on more than one covariance matrix at a time. (See Figure K-3.)

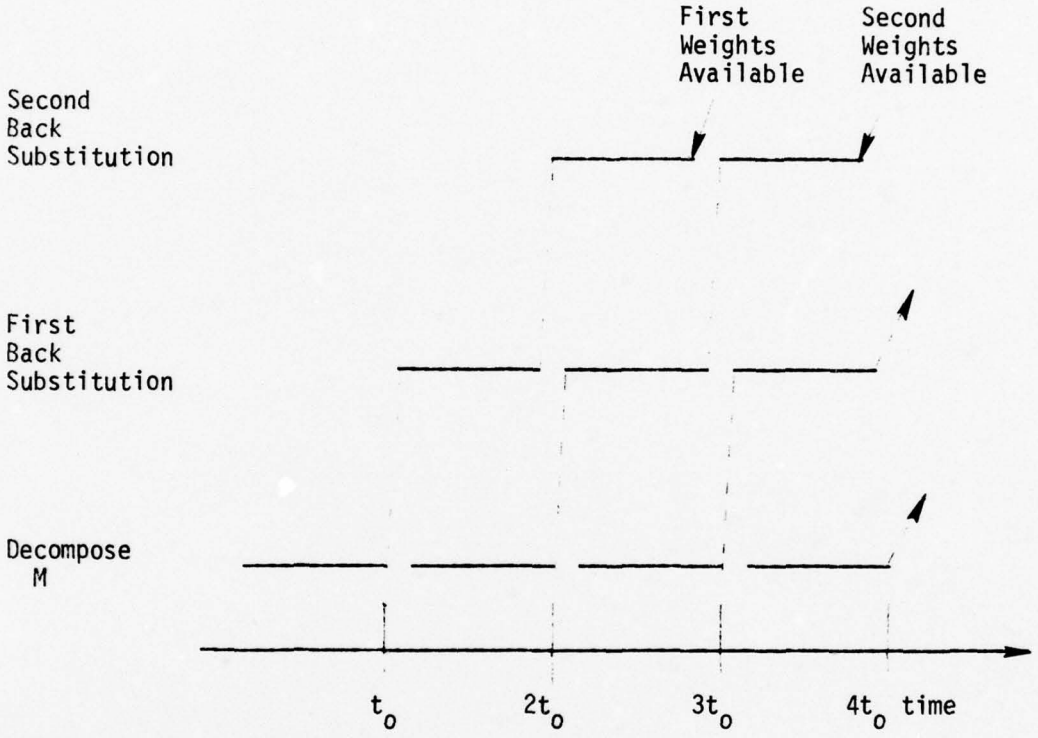


Figure K-3 Pipeline Decomposition and Back Substitution where $t_o = O(N)$

Appendix L. Implementations of Algorithms to Solve for Adaptive Weights on the PEPE

All the implementations in this section are written in an easily understood, structured programming language. These implementations are presented for operation counts purposes only and are not intended to be compilable code. For a discussion of these implementations, see Section 4.4.2.2, "Implementations of Algorithms to Solve for Adaptive Weights on the PEPE."

The variables used in these implementations may be divided into two classes, depending on whether they are stored in sequential memory (S) or parallel memory (P). Parallel variables are distinguished from sequential ones by a "*" as a subscript. For example, the parallel variable R(*) refers to corresponding memory locations in each PE. The parallel array X(N,*) is stored in N memory locations in each PE. When a parallel assignment statement is executed, such as

$$X(1,*) = R(*)*X(2,*) \quad ,$$

the multiplication is performed in each currently enabled processor simultaneously.

Mixed sequential-parallel assignment statements are allowed. If a parallel variable is on the left side of the equal sign,

R(*) = mixed expression (arithmetic expression co taining both sequential and parallel variables),

any number of PE's may be active, and each distinct sequential variable in the mixed expression must be moved from sequential to parallel memory

via an S-P operation. If a sequential variable is on the left side of the equal sign,

H(K) = mixed expression,

exactly one PE must be enabled, and the value calculated by the mixed expression in that PE will be transferred to sequential memory by a P-S operation.

The statements "ENABLE I" or "ENABLE I TO J" or "ENABLE $I_1,I_2,\ldots,I_n$" enable the indicated PE's and turn off all the others. The enabled PE's remain enabled until the next "ENABLE" statement is encountered.

Variables will either be of type integer (I), floating-point-real (R), or floating-point-complex (C). Complex variables are stored with their real and complex parts in separate locations, as described in the subsection of 4.4.2.2 entitled "Introduction," and are used to simplify notation. If it is necessary to refer to the real or imaginary parts of a complex variable separately, the notations REAL(·) and IMAG(·) will be used. (For example, REAL(M(J,*)) refers to the real part of the j^{th} row of M in all active parallel elements.) Similarly, CONJG(·) means the complex conjugate of the complex variable in parentheses.

Sequential variables used include N(type I, the dimension of the problem), K (type I, the number of steering vectors), NS (type I, the number of sample vectors), RECIP(N) (type R, the reciprocals of the diagonal elements of the factor matrix), and X(N) (type C, the sample vector). Parallel variables include the sample covariance matrix M(N,*)(type C), the conjugated steering vectors S(K,*) or S(N,*)(type C), and an array of the reciprocals

of the diagonal elements of the factor matrix, R(*) (type R--the i^{th} PE contains the reciprocal of the i^{th} diagonal element). Sample covariance matrix will be abbreviated to SCM.

Storage schemes are rowwise or columnwise for the (factored) SCM, and componentwise or vectorwise for the steering vectors, as discussed in the introduction to Section 4.4.2.2. The abbreviations for the storage schemes used in the back substitutions are given in Fig. 4.39c. For the first back substitutions, and second back substitutions with vectorwise steering vector storage, it is assumed the steering vectors are stored in the first K (componentwise storage) or N (vectorwise storage) PE's. The second back substitutions with componentwise steering vector storage may either follow an augmented or an unaugmented decomposition. In the first case, the steering vectors are stored next to M in PE's N+1 through N+K (see Fig. 4.39e), and in the second case they are stored below M in PE's 1 through K. This difference does not affect the operation counts of the implementation, just the means of accessing the data. To represent both schemes, we have chosen the following notation: in any line of code with the second part of the line set off in square brackets ([]), the code outside the brackets will refer to the augmented case, and the code inside brackets will refer to the unaugmented case. Similarly, bracketed variables in the variable list preceding the code itself are used in the unaugmented case only.

The first back substitutions solve LDT = $\overline{S}$ when M = LDL* (LT = $\overline{S}$ when M = LL*) and the second back substitutions solve L*W=T when M=LDL* (L*W=T when M=LL*).

Before each implementation is the number of PE's required and a list of all variables used, with their lengths (length per PE for parallel variables), locations (S or C), types (I, R, or C), and descriptions of their contents (omitted for M, S, RECIP, X, R).

FOR-END FOR loops behave as do those in Appendices A and B.

Method 1 for Updating the Sample Covariance Matrix
Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
X(N)	2N	S	C	sample vector
XT(*)	2	P	C	transpose of sample vector
JROW	1	S	I	current row of SCM being calculated

```
      BEGIN

C     REPEAT THIS CODE FOR ALL NS SAMPLE VECTORS
C     MOVE TRANSPOSE OF SAMPLE VECTOR TO PE'S

      FOR JROW = 1 TO N
      .   ENABLE I
      .   XT(*) = CONGJ(X(JROW))
      END FOR

C     UPDATE EACH ROW

      FOR JROW = 1 TO N
      .   ENABLE JROW TO N
      .   M(JROW,*) = M(JROW,*) + X(JROW) * XT(*)
      END FOR
      END
```

Method 2 for Updating the Sample Covariance Matrix
Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
X(N)	2N	S	C	sample vector
XT(*)	2	P	C	transpose of sample vector
JROW	1	S	I	current row of SCM being calculated

```
      BEGIN

C     REPEAT THIS CODE FOR ALL NS SAMPLE VECTORS
C     MOVE TRANSPOSE OF SAMPLE VECTOR TO PE'S

      FOR JROW = 1 TO N
      .   ENABLE JROW
      .   XT(*) = CONGJ(X(JROW))
      END FOR

C     UPDATE EACH ROW

      ENABLE 1 TO N
      DO JROW = 1 TO N
      .   M(JROW,*) = M(JROW,*) + X(JROW)*XT(*)
      END DO
      END
```

Method 3 for Updating the Sample Covariance Matrix

Number of PE's required = $N \cdot (N+1)/2$

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	initially upper triangle of M stored rowwise in $N \cdot (N+1)/2$ processors, finally as seen in Fig. 4.39a but upper half only
X(N)	2N	S	C	sample vector
X2(*)	2	P	C	1^{st} to N^{th} conjugated components of sample vector followed by 2^{nd} to N^{th} conjugated components of sample vector, 3^{rd} to N^{th}, etc., in all $N \cdot (N+1)/2$ PE's
X1(*)	2	P	C	1^{st} component of sample vector N times, followed by 2^{nd} component of sample vector N-1 times, 3^{rd} component N-2 times, etc., in all $N \cdot (N+1)/2$ PE's
MM(·)	$N \cdot (N-1)$	S	C	used to store 2^{nd} through N^{th} rows of outer product of sample vector
ISTR	1	S	I	first PE to enable
ISTP	1	S	I	last PE to enable
ISUB	1	S	I	location in MM(M) of current element of SCM being moved to M(MM)
JROW	1	S	I	row of current element of SCM
JCOL	1	S	I	column of current element of SCM

```
      BEGIN

C     REPEAT THIS CODE FOR ALL NS SAMPLE VECTORS

      ISTR = 1
      ISTP = 1

C     MOVE SAMPLE VECTOR INTO PE'S

      FOR JROW = 1 TO N
      .   ENABLE ISTR TO ISTP
      .   X1(*) = X(JROW)
      .   ISTR = ISTP + 1
      .   ISTP = ISTP + N - JROW
      .   ENABLE J,J+N-1,J+2*N-3,J+3*N-6,...,J+K*N-K*(K+1)/2,
      .                                  ...,(J-1)*N-J*(J-3)/2
      .   X2(*) = X(JROW)
      END FOR
      ENABLE 1 TO N*(N+1)/2
      M(1,*) = M(1,*) + X1(*)*X2(*)

C
C     EXECUTE THIS CODE AFTER ALL NS SAMPLE VECTORS DONE

      FOR ISUB = N+1 TO N*(N+1)/2
      .   ENABLE ISUB
      .   MM(ISUB-N) = M(1,*)
      END FOR
      FOR JROW = 2 TO N
      .   ENABLE JROW
      .   ISUB = N - 1 + JROW
      .   FOR JCOL = 2 TO JROW
      .   .   M(JCOL,*) = MM(ISUB)
      .   .   ISUB = ISUB + N - JCOL
      .   END FOR
      END FOR
      END
```

Gauss-Jordan (GJ)

Number of PE's required = N+K

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	augmented (see Fig. 4.39e)
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP(N)	2N	S	C	elements of JCOLth column of M
JROW	1	S	I	current row being eliminated
JCOL	1	S	I	current column being zeroed out

```
      BEGIN
C     ELIMINATE EACH COLUMN
      FOR JCOL = 1 TO N
      .   ENABLE JCOL
      .   R(*) = 1./REAL(M(JCOL,*))
      .   RECIP(JCOL) = R(*)
      .   FOR JROW = 1 TO N EXCEPT JCOL
      .   .   TEMP(JROW) = M(JROW,*)
      .   END FOR
      .   ENABLE 1 TO N+K
C     *   NORMALIZE THE JCOL-TH ROW
      .   M(JCOL,*) = RECIP(JCOL) * M(JCOL,*)
C     *   ELIMINATE THE JCOL-TH COLUMN
      .   FOR JROW = 1 TO N EXCEPT JCOL
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JROW) * M(JCOL,*)
      .   END FOR
      END FOR
      END
```

GE--*unaugmented*

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP(N)	2N	S	C	contains JCOLth column of SCM
JROW	1	S	I	current row being eliminated
JCOL	1	S	I	current column being zeroed out

```
      BEGIN

C     ELIMINATE EACH COLUMN

      FOR JCOL = 1 TO N-1
      .   ENABLE JCOL
      .   R(*) = 1./REAL(M(JCOL,*))
      .   RECIP(JCOL) = R(*)
      .   FOR JROW = JCOL+1 TO N
      .   .   TEMP(JROW) = M(JROW,*)
      .   END FOR
      .   ENABLE JCOL TO N

C     *   NORMALIZE JCOL-TH ROW

      .   M(JCOL,*) = RECIP(JCOL) * M(JCOL,*)

C     *   ELIMINATE JCOL-TH COLUMN

      .   FOR JROW = JCOL+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JROW) * M(JCOL,*)
      .   END FOR
      END FOR

C     CALCULATE LAST RECIPROCAL DIAGONAL

      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)
      END
```

LDL*--unaugmented--optimized

This implementation is identical to GE--unaugmented.

LDL*--unaugmented--unoptimized

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
R(*)	1	P	R	reciprocal diagonal
T(*)	2	P	C	temporary containing unnormalized JROWth row of SCM
RECIP(N)	N	S	R	reciprocal diagonals
TEMP	2	S	C	temporary containing the (JROW,JSUB)th element of SCM
JROW	1	S	I	current row of SCM being calculated
JSUB	1	S	I	row from which multiple of JROWth row is subtracted

```
      BEGIN

C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS

      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   R(*) = 1./REAL(M(JROW,*))
      .   RECIP(JROW) = R(*)
      .   ENABLE JROW+1 TO N

C     *   NOMALIZE JROW-TH ROW

      .   T(*) = M(JROW,*)
      .   M(JROW,*) = RECIP(JROW) * M(JROW,*)

C     *   SUBTRACT MULTIPLES OF JROW-TH ROW FROM FOLLOWING ROWS

      .   FOR JSUB = JROW+1 TO N
      .   .   ENABLE JSUB
      .   .   TEMP = T(*)
      .   .   ENABLE JSUB TO N
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP * M(JROW,*)
      .   END FOR
      END FOR
      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)
      END
```

LL*--unaugmented--optimized

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP(N)	2N	S	C	contains JCOLth column of SCM
SQR(*)	1	P	R	square roots of diagonal elements
SQ(N)	N	S	R	square roots of diagonal elements
JROW	1	S	I	current row being eliminated
JCOL	1	S	I	current column being zeroed out

```
      BEGIN

C     THIS IMPLEMENTATION IS IDENTICAL TO GE-UNAUGMENTED
C     EXCEPT FOR SOME ADDITIONAL CODE AT END
C     ELIMINATE EACH COLUMN

      FOR JCOL = 1 TO N-1
      .   ENABLE JCOL
      .   R(*) = 1./REAL(M(JCOL,*))
      .   RECIP(JCOL) = R(*)
      .   FOR JROW = JCOL+1 TO N
      .   .   TEMP(JROW) = M(JROW,*)
      .   END FOR
      .   ENABLE JCOL TO N

C     *   NORMALIZE JCOL-TH ROW

      .   M(JCOL,*) = RECIP(JCOL) * M(JCOL,*)

C     *   ELIMINATE JCOL-TH COLUMN

      .   FOR JROW = JCOL+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JROW) * M(JCOL,*)
      .   END FOR
      END FOR

C     CALCULATE LAST RECIPROCAL DIAGONAL

      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)

C     CALCULATE SQUARE ROOTS OF DIAGONAL ELEMENTS
C     AND EACH ROW OF MATRIX BY CORRESPONDING VALUE

      ENABLE 1 TO N
      SQR(*) = 1./SQRT(R(*))
      FOR JROW = 1 TO N
      .   ENABLE J
      .   SQ(J) = SQR(*)
      END FOR
      ENABLE 1 TO N
      FOR JROW = 1 TO N
      .   M(JROW,*) = SQ(JROW) * M(JROW,*)
      END FOR
      END
```

LL*--unaugmented--unoptimized

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39a
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP	2	S	C	temporary containing (JROW,JSUB)th element of SCM
JROW	1	S	I	current row of SCM being calculated
JSUB	1	S	I	row from which multiple of JROWth row is subtracted

```
      BEGIN
C     SUBTRACT MULTIPLE OF EACH ROW FROM FOLLOWING ROWS
      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   R(*) = 1./SQRT(REAL(M(JROW,*)))
      .   RECIP(JROW) = R(*)
      .   ENABLE JROW TO N
C     *   CALCULATE JROW-TH ROW
      .   M(JROW,*) = RECIP(JROW) * M(JROW,*)
C     *   SUBTRACT MULTIPLES OF JROW-TH ROW FROM FOLLOWING ROWS
      .   FOR JSUB = JROW+1 TO N
      .   .   ENABLE JSUB
      .   .   TEMP = M(JROW,*)
      .   .   ENABLE JSUB TO N
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP * M(JROW,*)
      .   END FOR
      END FOR
      ENABLE N
      R(*) = 1./SQRT(REAL(M(N,*)))
      RECIP(N) = R(*)
      END
```

GE--augmented

Number of PE's required = N+K

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP(N)	2N	S	C	contains JCOLth column of SCM
JROW	1	S	I	current row being eliminated
JCOL	1	S	I	current column being zeroed out

```
      BEGIN

C     THIS IMPLEMENTATION IS ALMOST IDENTICAL TO
C     UNAUGMENTED GE EXCEPT 'ENABLE JCOL TO N'
C     BECOMES 'ENABLE JCOL TO N+K' AND THE LAST
C     ROW IS MULTIPLIED BY THE LAST RECIPROCAL DIAGONAL
C
C     ELIMINATE EACH COLUMN

      FOR JCOL = 1 TO N-1
      .   ENABLE JCOL
      .   R(*) = 1./REAL(M(JCOL,*))
      .   RECIP(JCOL) = R(*)
      .   FOR JROW = JCOL+1 TO N
      .   .   TEMP(JROW) = M(JROW,*)
      .   END FOR
      .   ENABLE JCOL TO N+K

C     *   NORMALIZE JCOL-TH ROW

      .   M(JCOL,*) = RECIP(JCOL) * M(JCOL,*)

C     *   ELIMINATE JCOL-TH COLUMN

      .   FOR JROW = JCOL+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JROW) * M(JCOL,*)
      .   END FOR
      END FOR

C     CALCULATE LAST RECIPROCAL DIAGONAL

      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)

C     MULTIPLY LAST ROW BY LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K
      M(N,*) = RECIP(N) * M(N,*)
      END
```

LDL*--augmented--optimized

This implementation is identical to GE--augmented.

LDL*--augmented--unoptimized

Number of PE's required = N+K

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e
R(*)	1	P	R	reciprocal diagonal
T(*)	2	P	C	temporary containing unnormalized JROWth row of SCM
RECIP(N)	N	S	R	reciprocal diagonals
TEMP	2	S	C	temporary containing the (JROW,JSUB)th elements of SCM
JROW	1	S	I	current row of SCM being calculated
JSUB	1	S	I	row from which multiple of JROWth row is subtracted

```
      BEGIN

C     THIS IMPLEMENTATION IS ALMOST IDENTICAL TO
C     LDL*-UNELUCIDATED-UNOPTIMIZED, EXCEPT
C     "ENABLE X TO N" BECOMES "ENABLE X TO N+K" AND
C     THE LAST ROW IS MULTPLIED BY THE LAST RECIPROCAL
C     DIAGONAL
C
C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS

      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   R(*) = 1./REAL(M(JROW,*))
      .   RECIP(JROW) = R(*)
      .   ENABLE JROW+1 TO N+K

C     *   NORMALIZE JROW-TH ROW

      .   T(*) = M(JROW,*)
      .   M(JROW,*) = RECIP(JROW) * M(JROW,*)

C     *   SUBTRACT MULTIPLES OF JROW-TH ROW FROM
C     *   FOLLOWING ROWS

      .   FOR JSUB = JROW+1 TO N
      .   .   ENABLE JSUB
      .   .   TEMP = T(*)
      .   .   ENABLE JSUB TO N+K
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP * M(JROW,*)
      .   END FOR
      END FOR
      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)

C     MULTIPLY THE LAST ROW BY THE LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K
      M(N,*) = M(N,*) * RECIP(N)
      END
```

LL*--augmented--optimized

Number of PE's required = N+K

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP(N)	2N	S	C	contains JCOLth column of SCM
SQR(*)	1	P	R	square roots of diagonal elements
SQ(N)	N	S	R	square roots of diagonal elements
JROW	1	S	I	current row being eliminated
JCOL	1	S	I	current column being zeroed out

```
      BEGIN

C     THIS IMPLEMENTATION IS ALMOST IDENTICAL TO
C     LL*-UNAUGMENTED-OPTIMIZED EXCEPT
C     "ENABLE JCOL TO N" BECOMES "ENABLE JCOL TO N+K"
C     AND MULTIPLYING THE LAST ROW BY THE LAST
C     RECIPROCAL DIAGONAL
C
C     ELIMINATE EACH COLUMN

      FOR JCOL = 1 TO N-1
      .   ENABLE JCOL
      .   R(*) = 1./REAL(M(JCOL,*))
      .   RECIP(JCOL) = R(*)
      .   FOR JROW = JCOL+1 TO N
      .   .   TEMP(JROW) = M(JROW,*)
      .   END FOR
      .   ENABLE JCOL TO N+K

C     *   NORMALIZE JCOL-TH ROW

      .   M(JCOL,*) = RECIP(JCOL) * M(JCOL,*)

C     *   ELIMINATE JCOL-TH COLUMN

      .   FOR JROW = JCOL+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JROW) * M(JCOL,*)
      .   END FOR
      END FOR

C     CALCULATE LAST RECIPROCAL DIAGONAL

      ENABLE N
      R(*) = 1./REAL(M(N,*))
      RECIP(N) = R(*)

C     MULTIPLY THE LAST ROW BY THE LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K
      M(N,*) = M(N,*) * RECIP(N)

C     CALCULATE SQUARE ROOTS OF DIAGONAL ELEMENTS
C     AND MULTIPLY EACH ROW OF MATRIX BY CORRESPONDING VALUE

      ENABLE 1 TO N
      SQR(*) = 1./SQRT(R(*))
      FOR JROW = 1 TO N
      .   ENABLE J
      .   SQ(J) = SQR(*)
      END FOR
      ENABLE 1 TO N+K
      FOR JROW = 1 TO N
      .   M(JROW,*) = SQ(JROW) * M(JROW,*)
      END FOR
      END
```

LL*-- augmented-- unoptimized

Number of PE's required = N+K

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e
R(*)	1	P	R	reciprocal diagonals
RECIP(N)	N	S	R	reciprocal diagonals
TEMP	2	S	C	temporary containing (JROW,JSUB)th element of SCM
JROW	1	S	I	current row of SCM being calculated
JSUB	1	S	I	row from which multiple of JROWth row is subtracted

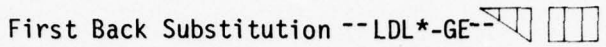
```
      BEGIN

C     THIS IMPLEMENTATION IS ALMOST IDENTICAL TO
C     LL*-UNAUGMENTED-UNOPTIMIZED EXCEPT
C     "ENABLE X TO N" BECOMES "ENABLE X TO N+K"
C     AND MULTIPLYING THE LAST ROW BY THE LAST
C     RECIPROCAL DIAGONAL
C
C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS

      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   R(*) = 1./SQRT(REAL(M(JROW,*)))
      .   RECIP(JROW) = R(*)
      .   ENABLE JROW TO N+K

C     *   CALCULATE JROW-TH ROW

      .   M(JROW,*) = RECIP(JROW) * M(JROW,*)

C     *   SUBTRACT MULTIPLES OF JROW-TH ROW FROM FOLLOWING ROWS

      .   FOR JSUB = JROW+1 TO N
      .   .   ENABLE JSUB
      .   .   TEMP = M(JROW,*)
      .   .   ENABLE JSUB TO N+K
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP * M(JROW,*)
      .   END FOR
      END FOR
      ENABLE N
      R(*) = 1./SQRT(REAL(M(N,*)))
      RECIP(N) = R(*)

C     MULTIPLY THE LAST ROW BY THE LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K
      M(N,*) = RECIP(N) * M(N,*)
      END
```

First Back Substitution--LDL*-GE-- 

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (rowwise)
S(K,*)	2K	P	C	see Fig. 4.39c (vectorwise)
TEMP(K)	2K	S	C	temporary containing JROW-1st components of S
R(*)	1	P	R	reciprocal diagonals
JROW	1	S	I	current component of S being calculated
AUG	1	S	I	current steering vector being calculated

```
! BEGIN

C     SUBTRACT MULTIPLES OF ROWS OF M FROM S

      FOR JROW = 2 TO N
      .   ENABLE JROW-1
      .   FOR AUG = 1 TO K
      .   .   TEMP(AUG) = S(AUG,*)
      .   END FOR
      .   ENABLE JROW TO N

C     *   CALCULATE JROW-TH COMPONENTS OF S

      .   FOR AUG = 1 TO K
      .   .   S(AUG,*) = S(AUG,*) - TEMP(J) * CONJG(M(JROW-1,*))
      .   END FOR
      END FOR

C     MULTIPLY BY RECIPROCAL DIAGONALS

      ENABLE 1 TO N
      FOR AUG = 1 TO K
      .   S(AUG,*) = S(AUG,*) * R(*)
      END FOR
      END
```

First Back Substitution --LDL*-GE-- 

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (rowwise)
S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)
TEMP(N)	2N	S	C	temporary containing JROWth column of L*
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of T being calculated
JSUB	1	S	I	component a multiple of which is being added to JROWth component

```
      BEGIN

C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS

      FOR JROW = 2 TO N
      .   ENABLE JROW
      .   FOR JSUB = 1 TO JROW-1
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE 1 TO K

C     *   CALCULATE JROW-TH COMPONENT OF S

      .   FOR JSUB = 1 TO JROW-1
      .   .   S(JROW,*) = S(JROW,*) - CONJG(TEMP(JSUB)) * S(JSUB,*)
      .   END FOR
      END FOR

C     MULTIPLY BY RECIPROCAL DIAGONALS

      FOR JROW= 1 TO N
      .   S(JROW,*) = S(JROW,*) * RECIP(JROW)
      END FOR
      END
```

First Back Substitution--LDL*-GE-- 

This implementation first transposes M and then uses First Back Substitution--LDL*-GE--  .

First Back Substitution--LDL*-GE-- 

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (columnwise)
S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)
TEMP(N)	2N	S	C	temporary containing JROWth row of L*
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of T
JSUB	1	S	I	component from which a multiple of the JROWth component is substituted

```
      BEGIN
C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS
      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   FOR JSUB = JROW+1 TO N
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE 1 TO K
C     *   CALCULATE JROW+1-ST COMPONENT OF S
      .   FOR JSUB =JROW+1 TO N
      .   .   S(JSUB,*) = S(JSUB,*) - CONJG(TEMP(JSUB)) * S(JROW,*)
      .   END FOR
      END FOR
C     MULTIPLY BY RECIPROCAL DIAGONALS
      FOR JROW = 1 TO N
      .   S(JROW,*) = RECIP(JROW) * S(JROW,*)
      END FOR
      END
```

First Back Substitution--LL*--

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (rowwise)
S(K,*)	2K	P	C	see Fig. 4.39c (vectorwise)
TEMP(K)	2K	S	C	temporary containing JROW-1st components of S
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of S being calculated
AUG	1	S	I	current steering vector being calculated

```
      BEGIN
C     SUBTRACT MULTIPLES OF ROWS OF M FROM S
      FOR JROW = 2 TO N
      .   ENABLE JROW-1
      .   FOR AUG = 1 TO K
C     *   *   MULTIPLY BY RECIPROCAL DIAGONAL
      .   .   S(AUG,*) = S(AUG,*) * RECIP(JROW-1)
      .   .   TEMP(AUG) = S(AUG,*)
      .   END FOR
      .   ENABLE JROW TO N
C     *   CALCULATE JROW-TH COMPONENTS OF S
      .   FOR AUG = 1 TO K
      .   .   S(AUG,*) = S(AUG,*) - TEMP(AUG) * CONJG(M(JROW-1,*))
      .   END FOR
      END FOR
C     MULTIPLY LAST COMPONENT BY LAST RECIPROCAL DIAGONAL
      ENABLE N
      FOR AUG = 1 TO K
      .   S(AUG,*) = S(AUG,*) * RECIP(N)
      END FOR
      END
```

First Back Substitution--LL*-- 

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (rowwise)
S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)
TEMP(N)	2N	S	C	temporary containing the JROWth column of L*
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of T being calculated
JSUB	1	S	I	component a multiple of which is being added to JROWth component

```
      BEGIN

C     SUBTRACT MULTIPLES OF EACH ROW FROM PRECEDING ROWS
C     CALCULATE FIRST COMPONENT OF S

      ENABLE 1 TO K
      S(1,*) = S(1,*) * RECIP(1)
      FOR JROW = 2 TO N
      .   ENABLE JROW
      .   FOR JSUB = 1 TO JROW-1
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE 1 TO K

C     *   CALCULATE JROW-TH COMPONENT OF S

      .   FOR JSUB = 1 TO JROW-1
      .   .   S(JROW,*) = S(JROW,*) - CONJG(TEMP(JSUB)) * S(JSUB,*)
      .   END FOR

C     *   MULTIPLY BY RECIPROCAL DIAGONAL

      .   S(JROW,*) = S(JROW,*) * RECIP(JROW)
      END FOR
      END
```

First Back Substitution--LL*-- 

This implementation first transposes M and then uses First Back Substitution--LL*--  .

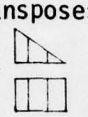
First Back Substitution--LL*-- 

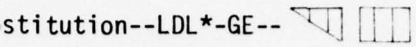
Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (columnwise)
S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)
TEMP(N)	2N	S	C	temporary containing JROWth row of L*
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of T
JSUB	1	S	I	component from which a multiple of the JROWth component is subtracted

```
      BEGIN

C     SUBTRACT MULTIPLES OF EACH ROW FROM FOLLOWING ROWS
C     CALCULATE FIRST COMPONENT OF S

      ENABLE 1 TO K
      S(1,*) = S(1,*) * RECIP(1)
      FOR JROW = 1 TO N-1
      .   ENABLE JROW
      .   FOR JSUB = JROW+1 TO N
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE 1 TO K

C     *   CALCULATE JROW+1-ST COMPONENT OF S

      .   FOR JSUB = JROW+1 TO N
      .   .   S(JSUB,*) = S(JSUB,*) - CONJG(TEMP(JSUB)) * S(JROW,*)
      .   END FOR

C     *   MULTIPLY BY RECIPROCAL DIAGONAL

      .   S(JROW+1,*) = S(JROW+1,*) * RECIP(JROW+1)
      END FOR
      END
```

Second Back Substitution--LDL*-GE-- 

This implementation transposes M first and then uses Second Back Substitution--LDL*-GE--  .

Second Back Substitution--LDL*-GE-- 

Number of PE's required = N+K [N]

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e (rowwise) [see Fig. 4.39b (rowwise)]
[S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)]
TEMP(N)	2N	S	C	temporary containing JROWth column of M
JROW	1	S	I	last calculated component of S
JSUB	1	S	I	component of S from which a multiple of the JROWth component is subtracted

```
      BEGIN
C     SUBTRACT MULTIPLES OF EACH COMPONENT FROM PRECEDING COMPONENTS
      FOR JROW = N TO 2 BY -1
      .   ENABLE JROW
      .   FOR JSUB = 1 TO JROW-1
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE N+1 TO N+K     [ 1 TO K ]
C     *   SUBTRACT MULTIPLE OF JROW-TH COMPONENT FROM PRECEDING COMPONENTS
      .   FOR JSUB = 1 TO JROW-1
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP(JSUB) * M(JROW,*)
     *.   .           [  S(JSUB,*) = S(JSUB,*) - TEMP(JSUB) * S(JROW,*) ]
      .   END FOR
      END FOR
      END
```

Second Back Substitution--LDL*-GE-- 

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (columnwise)
S(K,*)	2K	P	C	see Fig. 4.39c (vectorwise)
TEMP(N)	2N	S	C	temporary containing JROWth components of S
JROW	1	S	I	current row of M a multiple of which is subtracted from S
AUG	1	S	I	current steering vector

```
      BEGIN

C     SUBTRACT FROM THE COMPONENTS OF S MULTIPLES OF THE ROWS OF M

      FOR JROW = N TO 2 BY -1
      .   ENABLE JROW
      .   FOR AUG = 1 TO K
      .   .   TEMP(AUG) = S(AUG,*)
      .   END FOR
      .   ENABLE 1 TO JROW-1

C     *   DO EACH STEERING VECTOR

      .   FOR AUG = 1 TO K
      .   .   S(AUG,*) = S(AUG,*) - TEMP(AUG) * M(JROW,*)
      .   END FOR
      END FOR
      END
```

Second Back Substitution--LDL-GE-- 

Number of PE's required = N+K [N]

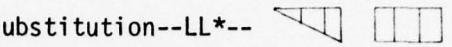
Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e (columnwise) [see Fig. 4.39b (columnwise)]
[S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)]
TEMP(N)	2N	S	C	temporary containing JROWth row of L
JROW	1	S	I	current component of S being calculated
JSUB	1	S	I	component of S a multiple of which is being subtracted from the JROWth component

```
      DEGIN

C     SUBTRACT FROM EACH COMPONENT MULTIPLES OF FOLLOWING COMPONENTS

      FOR JROW = N-1 TO 1 BY -1
      .   ENABLE JROW
      .   FOR JSUB = JROW+1 TO N
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE N+1 TO N+K     [ 1 TO K ]

C     *   CALCULATE JROW-TH COMPONENT

      .   FOR JSUB = JROW+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JSUB) * M(JSUB,*)
     *.   .             [ S(JROW,*) = S(JROW,) - TEMP(JSUB) * S(JSUB,*) ]
      .   END FOR
      END FOR
      END
```

Second Back Substitution--LL*-- 

Number of PE's required = N

This implementation transposes M first and then uses Second Back Substitution--LL*--  .

Second Back Substitution--LL*-- 

Number of PE's required = N+K [N]

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e (rowwise) [see Fig. 4.39b (rowwise)]
[S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)]
TEMP(N)	2N	S	C	temporary containing JROWth column of M
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	last calculated component of S
JSUB	1	S	I	component of S from which a multiple of the JROWth component is subtracted

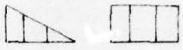
```
      BEGIN

C     SUBTRACT MULTIPLES OF EACH COMPONENT FROM PRECEDING COMPONENTS
C     MULTIPLY LAST COMPONENT BY LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K      [ 1 TO K ]
      M(N,*) = M(N,*) * RECIP(N)    [ S(N,*) = S(N,*) * RECIP(N) ]
      FOR JROW = N TO 2 BY -1
      .   ENABLE JROW
      .   FOR JSUB = 1 TO JROW-1
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE N+1 TO N+K     [ 1 TO K ]

C     *   SUBTRACT MULTIPLES OF JROW-TH COMPONENT FROM PRECEDING COMPONENTS

      .   FOR JSUB = 1 TO JROW-1
      .   .   M(JSUB,*) = M(JSUB,*) - TEMP(JSUB) * M(JROW,*)
     *.   .         [ S(JSUB,*) = S(JSUB,*) - TEMP(JSUB) * S(JROW,*) ]
      .   END FOR

C     *   MULTIPLY BY RECIPROCAL DIAGONAL

      .   M(JROW-1,*) = M(JROW-1,*) * RECIP(JROW-1)
     *.         [ S(JROW-1,*) = S(JROW-1,*) * RECIP(JROW-1) ]
      END FOR
      END
```

Second Back Substitution--LL*--

Number of PE's required = N

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39b (columnwise)
S(K,*)	2K	P	C	see Fig. 4.39c (vectorwise)
TEMP(N)	2N	S	C	temporary containing JROWth components of S
R(*)	1	P	R	reciprocal diagonals
JROW	1	S	I	current row of M a multiple of which is subtracted from S
AUG	1	S	I	current steering vector

```
      BEGIN

C     SUBTRACT FROM THE COMPONENTS OF S MULTIPLES OF ROWS OF M

      FOR JROW = N TO 2 BY -1
      .   ENABLE JROW
      .   FOR AUG = 1 TO K
      .   .   S(AUG,*) = S(AUG,*) * R(*)
      .   .   TEMP(AUG) = S(AUG,*)
      .   END FOR
      .   ENABLE 1 TO JROW-1

C     *   DO EACH STEERING VECTOR

      .   FOR AUG = 1 TO K
      .   .   S(AUG,*) = S(AUG,*) - TEMP(AUG) * M(JROW,*)
      .   END FOR
      END FOR

C     MULTIPLY FIRST COMPONENT BY FIRST RECIPROCAL DIAGONAL

      ENABLE 1
      FOR AUG = 1 TO K
      .   S(AUG,*) = S(AUG,*) * R(*)
      END FOR
      END
```

Second Back Substitution--LL*-- 

Number of PE's required = N+K [N]

Variable Name	Length	Location	Type	Description
M(N,*)	2N	P	C	see Fig. 4.39e(columnwise) [see Fig. 4.39b (columnwise)]
[S(N,*)	2N	P	C	see Fig. 4.39c (componentwise)]
TEMP(N)	2N	S	C	temporary containing JROWth row of L
RECIP(N)	N	S	R	reciprocal diagonals
JROW	1	S	I	current component of S being calculated
JSUB	1	S	I	component of S a multiple of which is being subtracted from the JROWth components

```
      BEGIN

C     SUBTRACT FROM EACH COMPONENT MULTIPLES OF FOLLOWING COMPONENTS
C     MULTIPLY LAST COMPONENT BY LAST RECIPROCAL DIAGONAL

      ENABLE N+1 TO N+K  [ 1 TO K ]
      M(N,*) = M(N,*) * RECIP(N)    [ S(N,*) = S(N,*) * RECIP(N) ]
      FOR JROW = N-1 TO 1 BY -1
      .   ENABLE JROW
      .   FOR JSUB = JROW+1 TO N
      .   .   TEMP(JSUB) = M(JSUB,*)
      .   END FOR
      .   ENABLE N+1 TO N+K     [ 1 TO K ]

C     *   CALCULATE JROW-TH COMPONENT

      .   FOR JSUB = JROW+1 TO N
      .   .   M(JROW,*) = M(JROW,*) - TEMP(JSUB) * M(JSUB,*)
     *.   .   [ S(JROW,*) = S(JROW,*) - TEMP(JSUB) * S(JSUB,*) ]
      .   END FOR

C     *   MULTIPLY BY RECIPROCAL DIAGONAL

      .   M(JROW,*) = M(JROW,*) * RECIP(JROW)
     *.       [ S(JROW,*) = S(JROW,*) * RECIP(JROW) ]
      END FOR
      END
```