

AD-A125 560

HIGH-PERFORMANCE BANDED EQUATION SOLVER FOR THE CRAY-1
II THE SYMMETRIC C... (U) MICHIGAN UNIV ANN ARBOR SYSTEMS
ENGINEERING LAB D A CALAHAN 01 OCT 82 SEL-166

1/1

UNCLASSIFIED

AFOSR-TR-83-0079 AFOSR-80-0158

F/G 12/1

NL

END

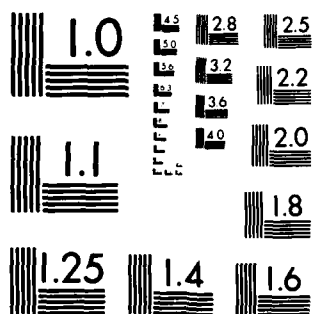
DATE

FILED

4 83

DTIC

M-2



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

High-Performance Banded Equation Solver for the CRAY-1

II. The Symmetric Case

D.A. CALAHAN

October 1, 1982

DTIC
A. ELECTRONICS
OCT 1 1982



Sponsored jointly by
Air Force Flight Dynamics Laboratory
Wright-Patterson Air Force Base
and Directorate of Mathematical and Information Sciences,
Air Force Office of Scientific Research,
under Grant No. ~~80-0158~~ *71561-80-158*

Systems Engineering Laboratory

DTIC FILE COPY

83 00 14 018

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0079	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) HIGH-PERFORMANCE BANDED EQUATION SOLVER FOR THE CRAY-1: II. THE SYMMETRIC CASE		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER SEL # 166
7. AUTHOR(s) D. A. Calahan		8. CONTRACT OR GRANT NUMBER(s) AFOSR 80-0158
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Michigan Dept. of Elec. & Computer Engring. Ann Arbor, MI, 48109		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 230A/A3
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research (NM) Bolling AFB, Washington DC, 20332		12. REPORT DATE October 1, 1982
		13. NUMBER OF PAGES 18
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
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)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Sparse matrices Parallel processing Vector processing Linear algebra		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes algorithms, performance, applications, and user information associated with a code which solves a memory-resident single banded matrix equation on the CRAY-1. The code is available as part of a library of CAL-coded equation-solvers for the CRAY-1		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

II. The Symmetric Case

AIRPORT SECURITY SERVICE (AFSSC)
NO 101
12.12.12.
Division

TABLE OF CONTENTS

	<u>Page</u>
Abstract	1
I. Introduction	2
II. Symmetric Banded Solution.	3
A. Basic Algorithm.	3
B. Partitioned Solution	5
Figure 1	6
Table 1.	7
III. Software Description	9
A. Storage Options.	9
B. Calling Sequences.	9
Figure 2	10
C. Comments	12
D. Driver Program	12
IV. Performance.	13
Table 2.	14
References	15
Appendix A	16

ABSTRACT

This report describes algorithms, performance, applications, and user information associated with a code which solves a memory-resident single banded symmetric matrix equation on the CRAY-1.

The code is available as part of a library of CAL-coded equation-solvers, [13].

PREFACE

The mathematical software described herein is the result of experimental research on vector algorithms for the direct solution of finite element grids arising in structural analysis. It represents what is thought to be the best compromise between vectorizability, sparsity exploitation, and user convenience for such problems for the CRAY-1.

I. Introduction

When direct methods are used in the solution of equations associated with 2-D finite element systems, the majority of production codes require a "frontal" approach [1], i.e., the finite elements are assembled and reduced in batches along a front that moves across the grid. This procedure saves storage of the entire profile matrix and so conserves memory, a major issue for the scalar scientific processors of the 1970's with fast storage often less than 100,000 words. Only relatively small research problems can be completely assembled and then completely solved in main memory. The principal difficulty with a frontal solution is programming complexity due to solution partitioning and to I/O management.

In contrast, vector processors with one- and two-megaword storage permit the memory-resident solution of the larger problems commensurate with their speed. Roughly, matrices associated with square grids three times larger on a side can now be solved by a vector processor, in the same computation time and at the storage limit of main memory.

A previous study [2] indicated that, for unsymmetric matrices, profile solution was marginally faster than banded solution on the CRAY-1. For this small speedup, significant preprocessing is required to block the profile structure. It was not considered worthwhile to produce a symmetric version of the block profile solution.

II. Symmetric Banded Solution

A. Basic algorithm

Consider an $n \times n$ symmetric banded matrix A , with half-bandwidth m . The solution of

$$AX = B$$

is performed in two steps, viz, (1) triangular factorization

$$A = U^T D U$$

where D is a diagonal matrix, U is an upper triangular matrix, and U^T is the transpose of U , and (2) forward and backward substitution

$$Y_1 = (U^T)^{-1} B$$

$$Y_2 = D^{-1} Y_1$$

$$X = U^{-1} Y_2$$

Asymptotically in n , factorization of a symmetric matrix requires $1/2$ the computation of an unsymmetric matrix; the substitution steps require the same computation.

The performance of the algorithm depends on the vector length for small bandwidths and on the data flow between the vector registers and main memory for all bandwidths. Mathematically, the average vector length is restricted to $1/2$ that of the unsymmetric case. The data flow is minimized - and performance optimized - when accumulation is made into a single row or column from previously-factored rows and columns; a poor algorithm creating excessive data flow would be the common one based on an outer product of a row or column.

A column-oriented accumulation suggested by Jordan [3] and modified in [2] could be used with reduced success in the symmetric case, due to the halved vector length. However, the accumulation

kernel discussed in [2] suffers, for small bandwidths, from being instruction-bound and, for large bandwidths, from a shift operation in the chained sequence of vector innerloop instructions. In contrast, the following algorithm kernel consists of simply a vector-matrix multiply, which can be made quite efficient.

The accumulation is represented as being made into a row rather than a column (Figure 1). The product of the row vector to the left of the main diagonal and the triangular matrix above the accumulator to the right of the main diagonal is then added to the accumulator row. However, this simple kernel is complicated by the need to maintain components of both U and DU or UD.

The organization of the accumulation kernel has the following form (see Figure 1a). Let

- Y be the accumulator row
- R be the row vector, initially stored as a column of DU above the pivot
- C (current column) be a column of U to be computed from and stored over R
- D be an appropriate segment of the diagonal, before the current pivot position
- M be the triangular matrix DU above Y and including R

then the reduction kernel has the form

$$T \leftarrow D^{-1}R$$

$$Y \leftarrow Y + TM$$

$$C \leftarrow T$$

The current column C is a column of U; the accumulator Y is a row of DU. T is a temporary vector and resides in a vector register.

An illustrative Fortran program incorporating this algorithm

is given in Table 1. This will be exercised later to obtain performance comparisons with the more efficient assembly code to follow.

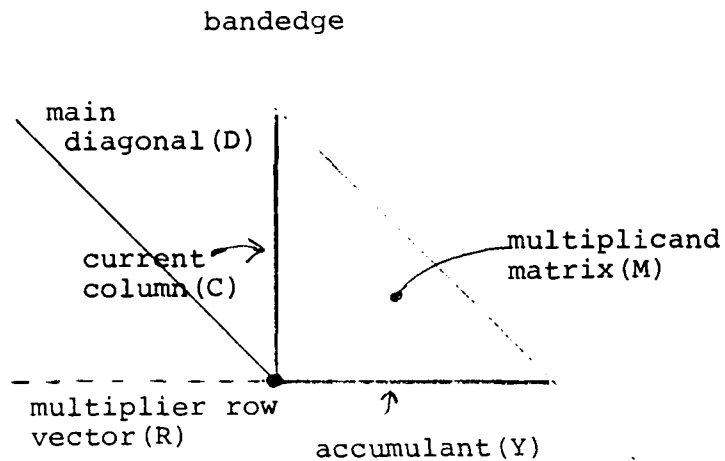
B. Partitioned Solution

When the half-bandwidth is greater than 64, so that vector lengths must be truncated, it remains efficient to preserve the concept of a vector-matrix multiply. The oversized matrix is then partitioned into bandedge and interior matrices, all of maximum dimension 64. Figure 1b illustrates this partitioning process in both the row and column directions. The circled numbers ①...④ represent the nesting level of the computation, with ① being the innermost loop. Loops ③ and ④ relate the order of the block processing; this ordering is also represented by the circled letters a...f.

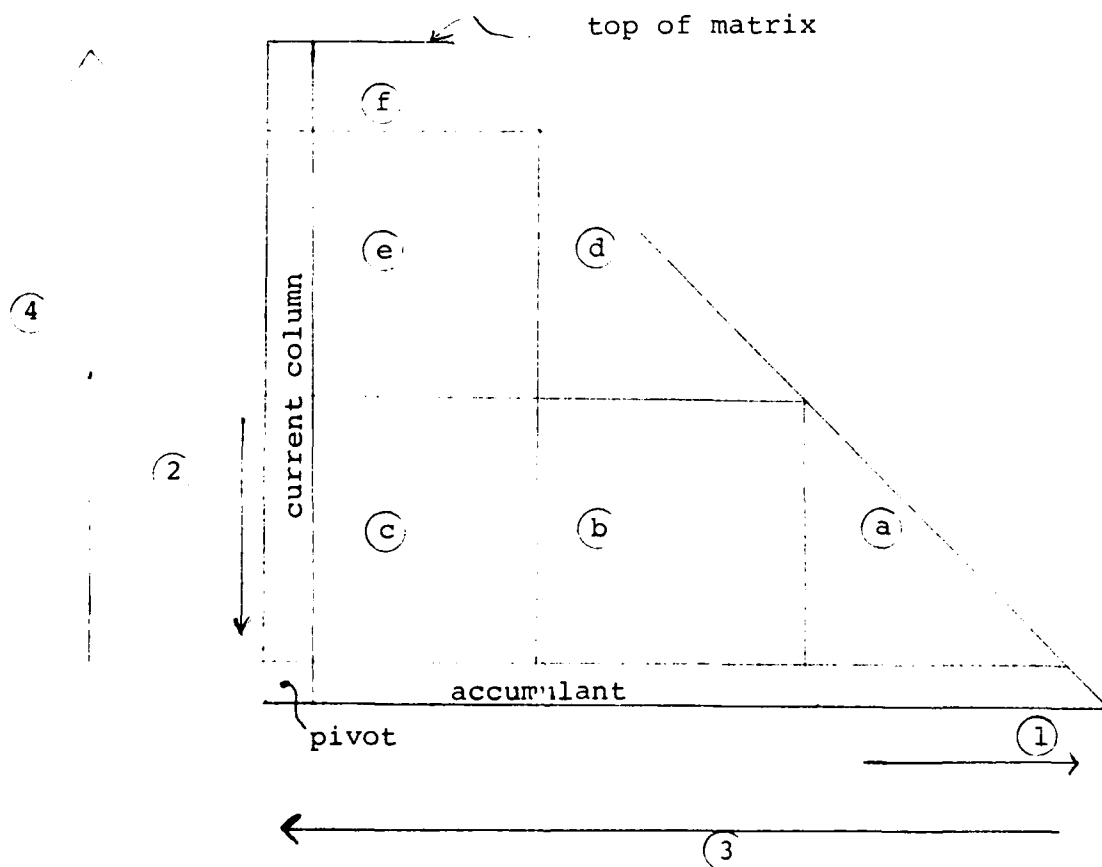
Two vector-matrix multiply kernels are now required: one for triangular bandedge matrices and one for rectangular (interior) matrices. The later kernels can achieve very high performance with short vector lengths. For vector lengths (VL) greater than 8, the execution rates are given by

$$\text{MFLOPS} = 160 \left(\frac{\text{VL}}{\text{VL}+8} \right)$$

i.e., a rate of 80 MFLOPS for VL = 8 and 142 MFLOPS for VL = 64.



(a) Overview



(b) Partitioned View

Figure 1. Organization and terminology of a reduction step

```

C**** ILLUSTRATIVE SYMMETRIC BANDED EQUATION SOLVER
C      UNPARTITIONED; ROW STORAGE ONLY
      DIMENSION A(1000), B(100), TEMP(100)
C**** M IS HALF-BANDWIDTH; N IS NUMBER OF EQUATIONS
10  READ (5,20,END=90) M, N
20  FORMAT (2I10)
      MP1 = M + 1
      M2 = 2 * M + 1
      DO 30 I = 1, N
30  B(I) = 0.
C**** FORMULATE EQUATIONS AND RHS TO HAVE SOLUTION B(J)=J
      DO 50 I = 1, MP1
        DO 50 J = 1, N
40      IX = I + (J - 1) * MP1
          A(IX) = 0.
          M1 = MAX0(1, M - I + 2)
          IF (J .GE. M1) A(IX) = -1.
          IF (I .EQ. MP1) A(IX) = M2
          IY = I + J - M - 1
          IF (IY .GT. 0 .AND. I .NE. MP1) B(J) = B(J) + (IY) * A(IX)
          IF (IY .GT. 0) B(IY) = B(IY) + (J) * A(IX)
50  CONTINUE
C**** TRIANGULARLY FACTOR MATRIX
      CALL FACTOR(A, TEMP, M, N)
C**** FORWARD AND BACK SUBSTITUTE
      CALL SOLVE(A, B, M, N)
      DO 60 J = 1, N
        AJ = J
        IF (ABS(B(J) - AJ) .GT. 1.E-6) GO TO 70
60  CONTINUE
      GO TO 10
70  WRITE (6,80) J, (B(I),I=1,N)
80  FORMAT (' FIRST WRONG SOLUTION VARIABLE IS', I5/(5E12.4))
90  STOP
      END
      SUBROUTINE FACTOR(A, TEMP, M, N)
      DIMENSION TEMP(1), A(1)
      MP1 = M + 1
      A(MP1) = 1.E0 / A(MP1)
      DO 50 J = 2, N
        JM1 = J - 1
        M1 = MAX0(1, J - M)
        ID = M1 * MP1 - MP1
        IX = M1 + J * M - 1
C**** IVDEP
      DO 10 I = M1, JM1
        IX = IX + 1
        ID = ID + MP1
10  TEMP(I) = A(IX) * A(ID)

```

Table 1. Simplified Fortran version of code

```

DO 30 I = M1, JM1
  IX = I - J + J * MP1
  US = TEMP(I)
  IZ = MINO(N, M + I)
  IL = I + J * M - M
  LJ = J * MP1 - M
CDIR$ IVDEP
  DO 20 L = J, IZ
    LJ = LJ + M
    IL = IL + M
  20  A(LJ) = A(LJ) - A(IL) * US
  30  CONTINUE
  IX = J * M + M1 - 1
CDIR$ IVDEP
  DO 40 I = M1, JM1
    IX = IX + 1
  40  A(IX) = TEMP(I)
    JJ = J * MP1
  50  A(JJ) = 1.E0 / A(JJ)
  RETURN
  END
  SUBROUTINE SOLVE(A, B, M, N)
  DIMENSION A(1), B(1)
  MP1 = M + 1
  NM1 = N - 1
  DO 10 I = 1, NM1
    IP1 = I + 1
    IL = I * MP1
    M1 = MINO(N, I + M)
  CDIR$ IVDEP
    DO 10 L = IP1, M1
      IL = IL + M
  10  B(L) = B(L) - A(IL) * B(I)
  CDIR$ IVDEP
    II = 0
    DO 20 I = 1, N
      II = II + MP1
  20  B(I) = B(I) * A(II)
    DO 30 L = 1, NM1
      LL = N - L + 1
      LLM1 = LL - 1
      M1 = MAXO(1, LL - M)
      ILL = M1 + LL * M - 1
  CDIR$ IVDEP
    DO 30 I = M1, LLM1
      ILL = ILL + 1
  30  B(I) = B(I) - A(ILL) * B(LL)
  RETURN
  END

```

Table 1. Continued

III. Software Description

A. Storage Options

It is common to store the diagonal and the U matrix in compressed form in an array of dimension $N*(M+1)^*$. Figure 2 illustrates eight possible regularly-addressed storage patterns; in each case, u_{ij} may be replaced by ℓ_{ji} to represent the storage of $L(=U^T)$ rather than U. Fortunately, all of these cases may be accommodated by defining suitable parameters of the argument lists of the following routines. The key to the generality is the passing of the (1,1) position of the matrix rather than the first element of the matrix storage array; all indexing is then performed off of this base.

B. Calling Sequences

Factorization

```
CALL SBANF (N,M,A(N11),NDIAG,NDROW)
```

where

N is the number of equations

M is the half-bandwidth (not including the diagonal)

A(N11) is the (1,1) element of the matrix

NDIAG is the storage increment between successive diagonal elements

NDROW is the storage increment between successive column elements

Substitution

```
CALL SBANS (N,M,A(N11),NDIAG,NDROW,Y)
```

* See following discussion for symbol definitions

```

      *
      *
      *
d1 u12 u13 u14
d2 u23 u24 u25
d3 u34 u35 u36
d4 u45 u46 *
d5 u56 * *
d6 * * *

```

(a) N11=1;NDROW=-(N-1);
NDCOL=N;NDIAG=1

```

      *
      *
      *
u14 u13 u12 d1
u25 u24 u23 d2
u36 u35 u34 d3
* u46 u45 d4
* * u56 d5
* * * d6

```

(c) N11=N*M+1;NDROW=N+1;
NDCOL=-N;NDIAG=1

```

d1 * * *
d2 u12 * *
d3 u23 u13 *
d4 u34 u24 u14
d5 u45 u35 u25
d6 u56 u46 u36
      * * *
      * *
      *

```

(b) N11=1;NDROW=-N;
NDCOL=N+1;NDIAG=1

```

* * * d1
* * u12 d2
* u13 u23 d3
u14 u24 u34 d4
u25 u35 u45 d5
u36 u46 u56 d6
* * *
* *
*

```

(d) N11=N*M+1;NDROW=N;
NDCOL=-(N-1);NDIAG=1

Figure 2. Permitted compressed storage; d_i is i th diagonal element; replace u_{ij} by u_{ji} when L is stored; $NDCOL = NDIAG - NDROW$.

```

* * * u14 u25 u36 * * *
* * u13 u24 u35 u46 * *
* u12 u23 u34 u45 u56 *
d1 d2 d3 d4 d5 d6

```

(e) N11=M+1; NDROW=1

NDCOL=M; NDIAG=N+1

(g) N11=1; NDROW=M;

NDCOL=1; NDIAG=M+1

```

* * * u14 u25 u36 * * *
* * u13 u24 u35 u46 * *
* u12 u23 u35 u45 u56 *
d1 d2 d3 d4 d5 d6

```

(f) N11=M+1; NDROW=M+2

NDCOL=-1; NDIAG=M+1

```

d1 d2 d3 d4 d5 d6
* u12 u23 u34 u45 u56 *
* * u13 u24 u35 u46 *
* * * u14 u25 u36 *

```

(h) N11=1; NDROW=-1

NDCOL=M+2; NDIAG=M+1

Figure 2. (Continued)

where

N...NDROW are defined above, except that A(N11) is the (1,1) element of the factorized matrix

Y is the right hand side on entry and the solution on exit.

C. Comments

1. Let $NDCOL = NDIAG - NDROW$ be the distance between successive elements in a row. When $|NDCOL|$ is a multiple of eight, performance of both the factorization and forward substitution step is severely degraded by a factor approaching four. When $|NDROW|$ or $|NDIAG|$ are multiples of eight, some degradation will also be noted for small bandwidths.
2. The dimension of Y must be at least $N + M + 1$.
3. The storage of the matrix may have to be increased to assure that certain data outside the normal matrix storage can be operated upon as floating point numbers. These positions are indicated by asterisks in Figure 1. For example, in Figure 1(a), the solver will access the data "above" the normal matrix storage, use it as operands for floating point add and multiply, but will not store the results. In this case additional storage need not be allocated, since these operands will simply be fetched from the preceding column. Only when this fetched data represents a fixed point or instruction format can floating point exceptions be expected.

D. Driver Program

Appendix A contains a listing of a Fortran driver program that formulates equations that are diagonally dominant and that are stored

so that neither $|NROW|$, $|NDCOL|$, or $|NDIAG|$ are multiples of eight, thus avoiding memory bank conflicts.

IV. Performance

Table 2 gives the measured solution times and execution rates associated with solving 1024 equations on the CRAY-1. Among the more interesting results are the rates for solving small-bandwidth cases, in comparison with the unsymmetric solver of [2] that has twice the average vector length. For half-bandwidths of 8, 16, and 32, the unsymmetric factorization executes at 18, 44, and 88 MFLOPS, respectively. From Table 2 the corresponding rates are 13.4, 34.0, and 68.9 MFLOPS. The asymptotic rates are similar for both solvers.

It should be pointed out that the timings in [2] were obtained on the COS operating system; CTSS was used to produce the results of Table 2.

Half-Bandwidth	Factorization	Substitution
4	.00508/5.03	.00122/14.2
8	.00616/13.4	.00122/27.6
16	.00865/34.0	.00122/54.1
32	.0160/68.9	.00201/64.7
64	.0401/105.	.00323/78.9
65	.0554/78.7	.00406/63.7
68	.0559/85.1	.00449/60.2
80	.0687/95.0	.00450/70.1
96	.0894/104.	.00468/80.2
128	.139/117.	.00552/89.2
129	.164/101.	.00624/79.4
132	.165/105.	.00667/75.9
144	.189/108.	.00674/81.4
160	.218/115.	.00686/88.1
196	.328/113.	.00886/82.0
197	.328/114.	.00886/82.3
200	.335/115.	.00886/83.5

Table 2 . Execution time (sec) and rate (MFLOPS) to solve 1024 equations

References

- [1]. Duff, I. S., and J. K. Reid, "Experience of Sparse Matrix Codes on the CRAY-1," Report CSS 116, AERE Harwell, October, 1981.
- [2]. Calahan, D. A., "High Performance Banded and Profile Equation Solvers for the CRAY-1: The Unsymmetric Case," Report #160, Systems Engineering Laboratory, University of Michigan, February, 1981.
- [3]. Jordan, T. and K. Fong, "Some Linear Algebraic Algorithms and Their Performance on the CRAY-1," Report LA-6774, Los Alamos Scientific Laboratory, June, 1977.

Appendix A
Listing of Driver Program

PAGE 1

DATE: 09-28-82, 10:59 OWNER: SMA FILE: SYMMETRIC.DR

```

1  C*** DRIVER FOR SYMMETRIC BANDSOLVER
2  DIMENSION A(550000), B(3000)
3  DO 96 JU=1,21
4  10 READ (5,20,END=100) M, N, NT
5  20 FORMAT (3I10)
6  MP1 = M + 1
7  M2 = 2 * M + 1
8  DO 30 I = 1, N
9  30 B(I) = 0.
10 C ROW STORAGE
11 NDIAG = MP1
12 NDROW = 1
13 N11 = MP1
14 IF (NT.EQ.0) GO TO 40
15 COLUMN STORAGE
16 NDIAG = 1
17 NDROW = -(N - 1)
18 N11 = 1
19 40 NTOT = MP1 * N
20 NDCOL = NDIAG - NDROW
21 IF ((NDCOL/8)*8-NDCOL.NE.0)GO TO 114
22 WRITE(6,77)
23 77 FORMAT(' EXECUTION WILL BE SLOWED BY BANK CONFLICTS;')
24 1' NDROW OR NDIAG WILL BE CHANGED')
25 IF (NDROW.NE.1)NDROW=-N-1
26 IF (NDIAG.NE.1)NDIAG=MP1+1
27 NDCOL=NDIAG-NDROW
28 114 DO 50 I=1,NTOT
29 50 A(I) = 0.
30 DO 60 J = 1, N
31 DO 60 I = 1, MP1
32 NCOL = J + I - 1
33 IF (NCOL.GT. N) GO TO 60
34 NA = N11 + NDIAG * (J - 1) + NDCOL * (I - 1)
35 A(NA) = -1.
36 IF (I.EQ.1) A(NA) = M2
37 B(J) = B(J) + NCOL * A(NA)
38 IF (I.NE.1) B(NCOL) = B(NCOL) + J * A(NA)
39 60 CONTINUE
40 T1=SECOND()
41 CALL SBANF(N, M, A(N11), NDIAG, NDROW)
42 T1=SECOND()-T1
43 T2=SECOND()
44 N2=N11+20
45 WRITE(6,91) (A(J),J=1,N2)
46 91 FORMAT(5E12.4)
47 CALL SBANS(N, M, A(N11), NDIAG, NDROW, B)
48 T2=SECOND()-T2
49 DO 70 J = 1, N
50 AJ = J
51 IF (ABS(B(J) - AJ).GT. 1.E-6) GO TO 80
52 70 CONTINUE
53 WRITE(6,92) M,N,NT
54 92 FORMAT(' M =',16,' N =',16,' NT =',16)
55 WRITE(6,93) T1,T2
56 93 FORMAT(' TIMINGS: ',2E16.6)
57 96 CONTINUE
58 GO TO 100

```

<PAGE 1>

//// FILE:SYMMETRIC.DR /////

>>>> MAIN PROGRAM <<<<

PAGE 1>

<PAGE 2>

| DATE: 09-28-82, 10:59 OWNER: SMXA FILE: SYMMETRIC.DR |

ISN

53
54
55
56

80 WRITE (6,90) J, (B(1),I=1,N)
90 FORMAT (' FIRST WRONG SOLUTION VARIABLE IS', 15/(5E12.4))
100 STOP
END

<PAGE 2>

59
60
61
62