General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

PACIFICA TECHNOLOGY

P.O. Box 148 • Del Mar, California 92014 • Tel: (714) 453-2530

PT-U79-0330

EVALUATION OF OUT-OF-CORE COMPUTER PROGRAMS FOR THE SOLUTION OF SYMMETRIC BANDED LINEAR EQUATIONS

by

Robert S. Dunham

Prepared for

George C. Marshall Space Flight Center
National Aeronautics and Space Administration

(NASA-CR-150708) EVALUATION OF OUT-OF-CORE COMPUTER PROGRAMS FOR THE SOLUTION OF SYMMETRIC BANDED LINEAR EQUATIONS (Pacifica Technology, Del Mar, Calif.) 161 p HC A08/MF A01 CSCL 09B G3/61

N78-24822

Unclas 21900

December 1976

ABSTRACT

This report evaluates fortran coded out-of-core equation solvers that solve using direct methods symmetric banded systems of simultaneous algebraic equations. The types of solvers studied were banded, frontal and column(skyline) solvers. Also considered were solvers that could "partition" the working area and thus could fit into any available core. Comparison timings are presented for several typical two-dimensional and three-dimensional continuum type grids of elements with and without mid-side nodes. Extensive conclusions are also given.

ACKNOWLEDGEMENTS

This report presents research primarily carried out at the University of Texas at Austin under contract to NASA-MSFC. The author gratefully acknowledges the cooperation and support of their computer center and the faculty and staff. Professors C. P. Johnson and E. B. Becker contributed substantially to both the work and understanding. The numerical experiments were performed by Chih-Shun Kenneth Wang as part of his M. S. Thesis work. Ricardo Nicolau is gratefully recognized for his assistance here also. Professor E. L. Wilson of the University of California at Berkeley is deserving of thanks for having willingly cooperated with the author in providing several of the codes used.

The author's present employer, Pacifica Technology, is also recognized for having supported the preparation of this report. Finally, John Key from MSFC should be recognized for having sponsored the effort.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

The purpose of this report is to evaluate fortran coded out-of-core equation solvers that solve using direct methods symmetric banded systems of simultaneous equations such as the equilibrium equations generated by static finite element or finite difference discretizations of two- and three-dimensional stress analysis problems. This research was carried out under contract to the N.A.S.A. Marshall Space Flight Center for the purpose of improving the operational efficiency of the TEXGAP computer program [1]. This report is directed to those individuals who develop, modify or routinely use finite element computer codes. No attempt is made herein to review finite element methodology and the reader is assumed to be familiar with fortran programming techniques and the techniques for solving systems of simultaneous equations on large digital computers.

There are 3 methods commonly used for solving systems of simultaneous equations on digital computers; direct methods, iterative methods and hybrid methods. Direct methods involve the reduction of the equations to an upper triangular form (this step is called forward substitution, reduction or triangularization) from which solution is effected by simple back-substitution. Iterative methods require an approximate starting solution from which successively better approximations are obtained from algebraic equations using one of many procedures such as Gauss-Seidel [2] or Successive Over-Relaxation [3]. Hybrid methods involve the combined use of direct and iterative methods.

The earliest finite element programs developed in the late 50's and early 60's used the Gauss-Seidel iteration method to solve for the nodal point displacements [4]. Iteration methods were probably selected by these early developers because they were structural analysts familiar with a similar iteration technique called "Moment Distribution". These early iteration schemes quickly gave way to the direct method of Gaussian elimination. Direct methods proved to be just as convenient to program and they were easier to use and gave accurate and reliable results for a large class of two-dimensional(2D) static linear elastic stress analysis problems.

At first, the direct equation solvers were programmed so that all the coefficients fit within the available core storage. Earlier research also funded by NASA studied these solvers [5]. At the time of this earlier research (the mid 60's), the available central memory core storage was generally either 32,000 or 65,000 words. Thus only 2-3,000 unknowns or degrees-of-freedom(DOF) could be handled using even the most efficient banded, symmetric solvers. As finite element methodology advanced, elements with more nodes and more DOF and three-dimensional(3D) applications were developed that required millions of words of storage, much more core than is available on any computer system. This lead to the development of solvers that partitioned the equations into smaller pieces that would fit into central memory core storage while the eliminations were being performed, and then these smaller pieces were written onto low speed tape, disk or drum storage to make room for the next set of coefficients.

While the techniques for programming Gaussian elimination in-core are straightforward and fairly well standardized, those used for out-of-core solvers are highly variable. Another complicating factor is that virtually every computer installation has a different method of charging for the transfer of data from central memory to low speed storage. Some make no charge and others may charge more for these input/output(I/O) transfers than for the actual central processor(CP) calculation time. For example, the University of Texas charges \$0.004 per each 64 words (called a Physical Record Unit) but converts this charge to equivalent CDC 6600 time at the rate of \$230 per hour. Thus, the charge rate amounts to 0.98×10^{-3} CP sec per word transferred. The importance of this charge can be seen in the following example. A system of $n=10^4$ equations at a bandwidth of $b=10^3$ requires approximately $nb^2/2 = 5 \times 10^9$ operations, i.e., the total number of multiplications and additions to triangularize the system. This will require approximately 2×10^4 CP seconds on a CDC 6600 assuming the average rate of computations is about $.25 \times 10^6$ operations per CP second. Provided that the entire bandwidth block fits in-core, there will be $2nb = 20 \times 10^6$ I/O transfers (1 write and 1 read of each coefficient) or an equivalent

charge of 2×10^4 CP seconds. Thus, the total charge will be 4×10^4 seconds, fully one-half being charged for I/O transfers.

The net effect of this is to make meaningful evaluation of out-of-core solvers very difficult because the concept of operational efficiency means performing a given set of computations for the minimum computational expense (i.e., the actual charge) and not the minimum central processor time. Further complications which are not considered herein also arise when consideration must be given to obtaining reasonable priority to achieve good turn-around.

In the next chapter is given background information on fortran coding techniques to solve systems of banded symmetric positive definite equations and specific attention is given to out-of-core banded, frontal and column solvers. Chapter 3 briefly describes some important characterics of the particular solvers used in the present study. Chapter 4 describes and presents the results of the various numerical experiments carried out to evaluate these solvers. Chapter 5 gives rather extensive conclusions and recommendations for future research.

CHAPTER 2 SOLUTION OF SIMULTANEOUS EQUATIONS BY ELIMINATION

2.1 Background/In-Core Solvers

A brief review of the basic techniques used in programming equation solvers will now be made to establish the terminology used in this report. In this section only in-core solvers are considered.

Let the simultaneous equations be representated as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad \text{for } i = 1,2,.....n \tag{2.1}$$

where a_{ij} are the stiffness coefficients, b_i the nodal point forces and x_j the unknowns(DOF), and n is the total number of DOF. The standard elimination procedure is to solve for x_1 in terms of x_2, x_3.....x_n from the first equation. It is important to use the first equation to eliminate the first unknown because no reordering (pivoting) is necessary*. Solving the first equation for x_1 gives

$$x_1 = \frac{1}{a_{11}}\left(b_1 - \sum_{j=2}^{n} a_{1j}x_j\right) \tag{2.2}$$

This equation is now substituted into equations 2 thru n

$$\frac{a_{i1}}{a_{11}}\left(b_1 - \sum_{j=2}^{n} a_{1j}x_j\right) + \sum_{j=2}^{n} a_{ij}x_j = b_i \tag{2.3}$$

and collecting

$$\sum_{j=2}^{n}\left(a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}\right)x_j = b_i - \frac{a_{i1}}{a_{11}}b_1 \tag{2.4}$$

for i = 2,3,.....n

*In spite of some controversy on this point, there is no recognized body of evidence that indicates reordering improves accuracy or reduces roundoff error with systems of linear equations derived from either displacement or mixed finite element models.

Thus, the order of the system of equations is reduced from size n to n-1. After n-1 such eliminations (x_2 from the second equation, etc.), the equations are reduced to an upper triangular form that permits solution for the unknowns by backsubstitution. This direct Gaussian elimination is straight forward to code in the Fortran language as is shown in Table 2.1 below.

```
      DO 300 K=1,N-1
      DO 200 I=K+1,N
      B(I)=B(I)-B(K)*A(I,K)/A(K,K)
      DO 100 J=K+1,N
  100 A(I,J)=A(I,J)-A(I,K)*A(K,J)/A(K,K)
  200 CONTINUE
  300 CONTINUE
C
C     BACKSUBSTITUTION
C
      X(N)=B(N)/A(N,N)
      I=N
  400 I=I-1
      SUM=0.0
      DO 500 J=I+1,N
  500 SUM=SUM-A(I,J)*X(J)
      X(I)=(B(I)-SUM)/A(I,I)
      IF (I.GT.1) GO TO 400
```

Table 2.1 Symbolic Fortran Routine for Direct Gaussian Elimination

The coding in Table 2.1 does not take advantage of either the symmetry or the banded nature of the equilibrium equations generated by finite element methods. As illustrated in Figure 2.1 only those coefficients above the main diagonal and within the band need be stored because the Fortran coding given in Table 2.1 will <u>not</u> cause any nonzero terms to appear outside the band

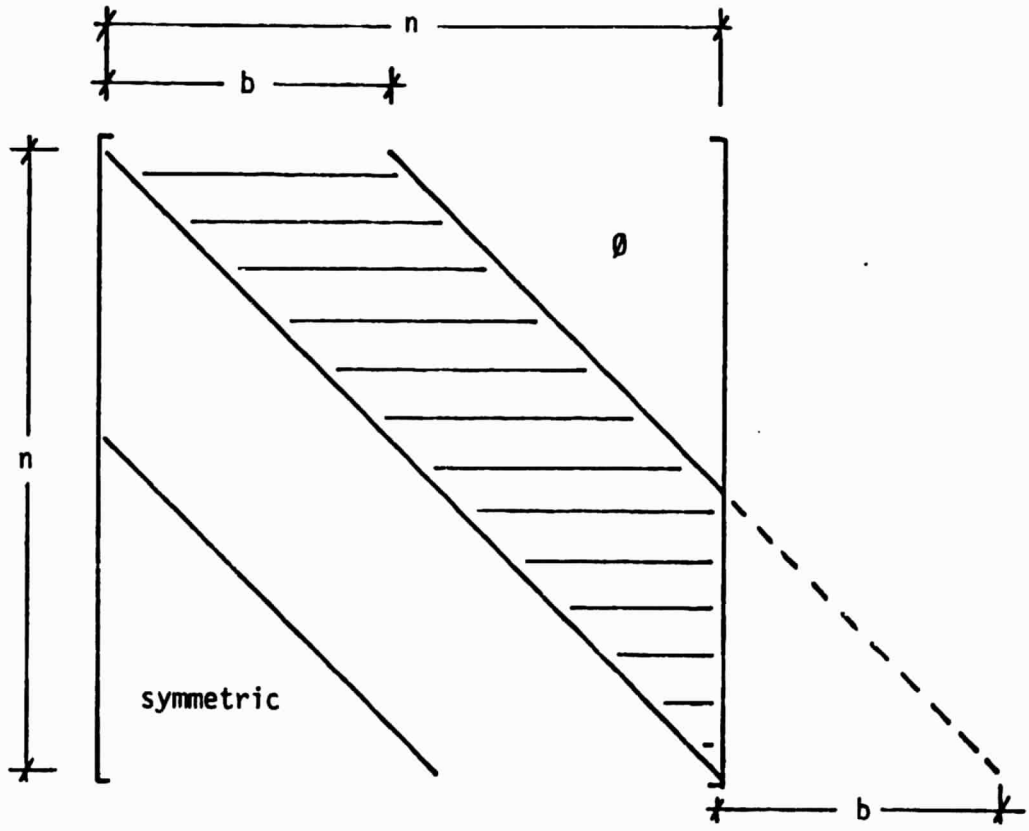


Figure 2.1 Symmetric Banded Equations

during the elimination process*. Thus, it is necessary to store only n*b (b=bandwidth) coefficients instead of the n^2 required for a general system. To take advantage of this storage reduction, it is necessary to store the equations in a tri-diagonal form in which column 1 of the tri-diagonal matrix gives the main diagonal of the actual system of equations, column 2 the first off-diagonal term and column b the edge of the band in any row. The Fortran coding to effect reduction of the equations in tri-diagonal form is given in Table 2.2. The coding in Table 2.2 is essentially the same as that in Table 2.1, however, advantage is taken of the banding and symmetry to reduce the number of passes through the innermost loop (300) from $n^3/3$ in the general case to $nb^2/2$.

```
      DO 300 K=1,N-1
      LIM=MIN(K+MBAND,N)
      DO 200 I=K+1,LIM
      B(I)=B(I)-A(K,I-K)*B(K)/A(K,1)
      DO 100 J=I,LIM
  100 A(I,J-I+1)=A(I,J-I+1)-A(K,I-K)*A(K,J-K)/A(K,1)
  200 CONTINUE
  300 CONTINUE
```

Table 2.2 Symbolic Fortran Routine for Tri-diagonalized Equations

There are many obvious Fortran improvements that can be made to the coding in Tables 2.1 and 2.2**, but these are not particularly relevant to the present study since the limiting feature of this coding is that it requires all coefficients to reside in-core during the solution. If we assume that the largest an array can be dimensioned is 4×10^4 decimal locations, this limits the size of the problem that can be solved to say

---

*Reordering might cause the band to increase.

**For example, the "DO" loops should be replaced by variable counters (I=I+1, etc.), the variable subscripting should be with single subscripts and the innermost sum accomplished with temporaries.

n=800 and b=50. This method of solution is then capable of handling one-dimensional(1D) continuum problems, 2D and some 3D truss and frame structures and some small 2D continuum problems (a plane strain grid with 400 nodes and a maximum bandwidth of 25 nodes). Clearly, in-core solvers have a very limited spectrum of applications.

2.2 Out-of-Core Solvers

It is very difficult to characterize out-of-core solvers since a very important feature is how the coefficients are stored and how many I/O transfers are required. Herein is briefly described three important types of out-of-core solvers; band, frontal and column solvers. For band and frontal solvers there is an important dichotomy between those that have sufficient storage available to handle the maximum band or front and those that lack sufficient storage and thus must subdivide the band or front. The subdivision of the band or front will be called "partitioning". This is not an entirely satisfactory name since partitioning is common used in matrix algebra to denote symbolic groupings, but it accurately conveys the intended meaning in the present study. Out-of-core column solvers are intrinsicly partitioned by the manner in which "blocking" is done. These factors will be discussed more fully in the next 3 sections.

2.2.1 Bandsolvers

All out-of-core bandsolvers use logic essentially the same as that given in Table 2.2 for effecting the elimination process, but they differ significantly in how they perform the I/O. Figures 2.2 and 2.3 illustrate the two extremes in terms of required core storage for bandsolvers that do not partition the band.

The sliding block scheme illustrated in Figure 2.2 requires storage for $b^2/2$ coefficients and as a row is eliminated it is written to low speed storage and the equations are shifted up one position [6]. There are several drawbacks to this method:

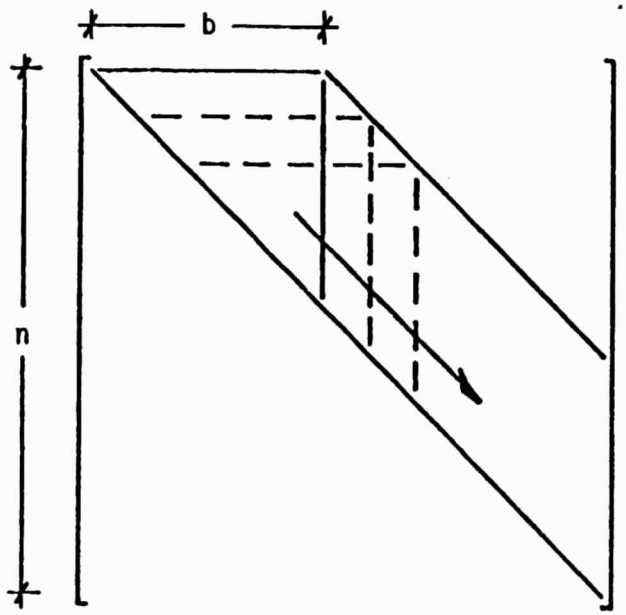


Figure 2.2 Sliding Triangular Block Band Solver

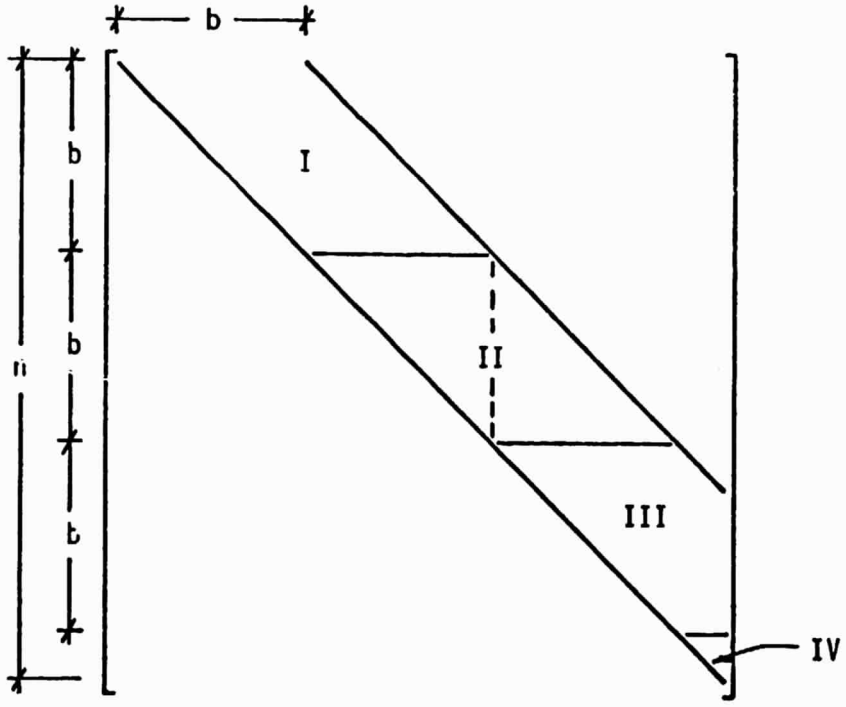


Figure 2.3 Bandwidth Block Band Solver

(1) I/O is required row-wise and therefore transfers small records,

(2) shifting takes place after each elimination which is a very wasteful procedure,

(3) the procedure for assembly and also that for bringing new columns into core are very complicated, and

(4) assembly and elimination cannot be performed simultaneously.

The bandwidth block scheme illustrated in Figure 2.3 avoids all of the above inefficiencies and complexities at the expense of storage. A total of $2b^2$ words are necessary because the first "b" unknowns are eliminated before any I/O is done. This requires that the triangular portion in block II be present during elimination of block I. Actually, only 1/2 of block II is needed but it is not generally convenient to take advantage of this space* so $b^2/2$ of storage is wasted. A significant advantage to the bandwidth block procedure is that it permits (in fact encourages) the combination of the assembly of the global stiffness from the element stiffness and the forward elimination process. This is significant because it eliminates wasted I/O and limits the total transfers to n*b words (1 word for each coefficient).

There are of course many other methods, but their differences are minor for purposes of the present study.

Without partitioning it is necessary to fit some multiple (2) of b^2 into central memory, thus if only 32,000 words are available then $b \leq \sqrt{\frac{32000}{2}} = 126$. However, on CDC 7600s and many other large computers there is often 250,000 words available. In this case $b \leq 353$, a very considerable number.

For large bandwidths it is necessary to partition the band as illustrated in Figure 2.4 for 3 partitions of the b^2 block I, I_1, I_2 & I_3 respectively. This procedure implies core residence of 2 of the $b^2/3$ blocks at any time. Blocks I_1 & I_2 initially are brought into core and

---

*Simply because the coding becomes too complicated.

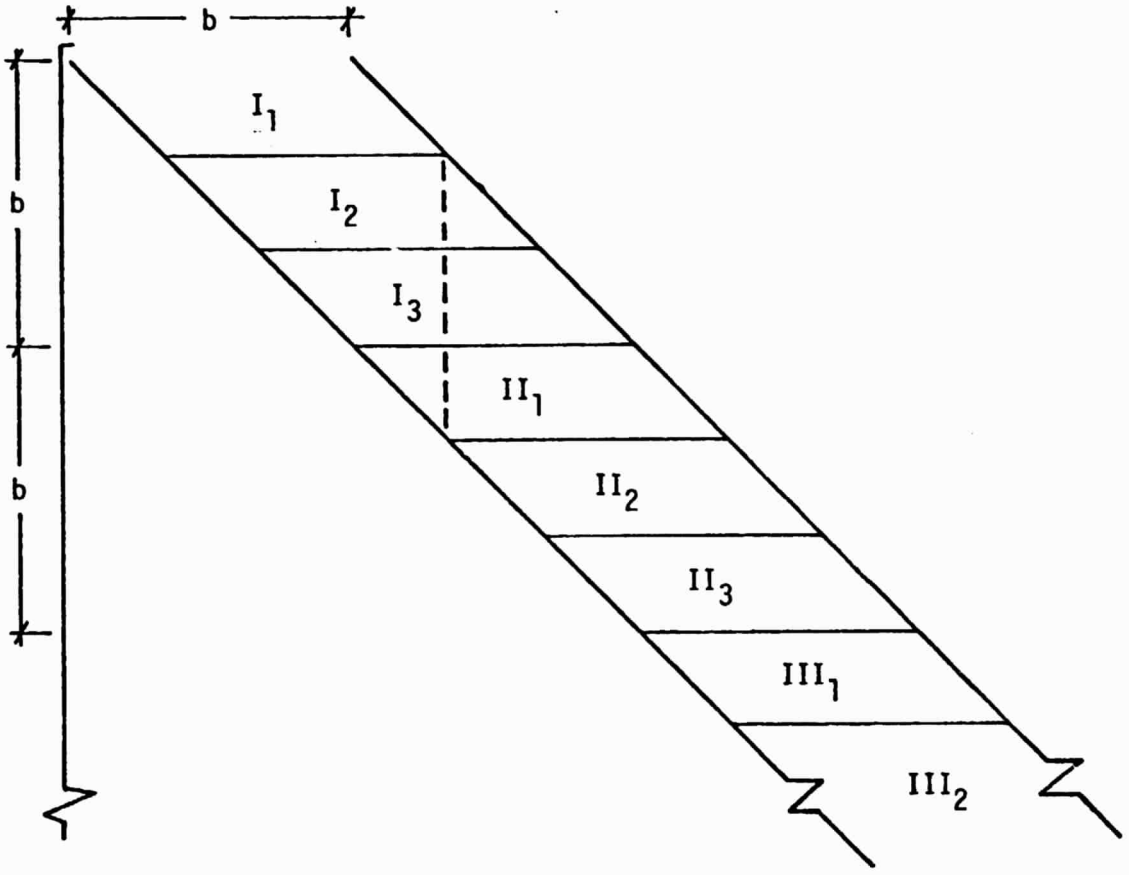


Figure 2.4 Typical Partitioning of the Band

I_1 is reduced through the end of I_2, then partially reduced block I_2 is written out and I_3 is brought in and the elimination of I_1 is made on it. I_3 is written and II_1 is treated the same as I_3. I_1 is now complete and written and partially reduced I_2 and I_3 are brought back in and so on. Table 2.3 gives the total number of reads and writes (1 for each) for only the reduction.

block	total number of reads & writes
I_1	3
I_2	5
I_3	7
II_1	9
II_2	9
II_3	9
III_1	9
III_2	7
	58

Table 2.3 Block I/Os in Reduction

Thus for this simple example, it is found that 58 I/Os of block size $b^2/3$ were required. Without partitioning there would have been 3 outputs of size b^2. The partioning scheme thus required $58b^2/3 \div 3b^2 = 58/9 \simeq 6$ times as many I/O word transfers and $58/3 \simeq 19$ times as many I/O accesses. Note that block I must be written during the assembly process and then read and written once during the reduction process. Assembly and reduction cannot be combined in a bandsolver that requires partitioning. Obviously, partitioning should be avoided whenever possible.

2.2.2 Frontal Solvers

Frontal solvers are fundamentally quite different than either band or column solvers (which are similar) in that the nodes or DOF are assigned location numbers in the active front based on their appearance in the element

being assembled. Thus, the ordering of the element stiffness matrices passed to a frontal solver determines the size of the front. The nodal numbers are used only as reference numbers or labels with which to compare with other labels (e.g., A1, B10, etc. are just as useful as 1, 210, etc.).

To illustrate how a frontal procedure works consider the grid shown in Figure 2.5. As element 1 is assembled, nodes 1,2,3,8,9,12,13 & 14 are given (column or front) locations 1,2,3,4,5,6,7 & 8, respectively in the "active" area of the band which is called the front. Since nodes 1,2 & 8 are connected only to element 1 they are now eliminated from the active area in the usual manner and the reduced coefficients passed to an internal buffer area that is dumped to low speed storage when full. This leaves the active area with a substructure stiffness consisting of the line of nodes 12-13-14-9-3. Next element 2 is assembled with <u>new</u> nodes 4,5,10,15 & 16. The new nodes are assembled into the first available positions in the front; 1,2,4,9 & 10, respectively. Nodes 3,9 & 14 are currently located in the front at locations 3,5 & 8, respectively. Since nodes 3,4 & 9 are complete they are now eliminated leaving the active front the nodal line 12-13-14-15-16-10-5. Table 2.4 summarizes the frontal locations and maximum front widths for all 6 elements. Zeroes (0) indicate that the node has not yet appeared, -1 indicates it has been previously eliminated, and 4* means that the node is in position 4 and will be eliminated because the current element is the last appearance of that node. Note that after an initial transient the front remains steady for regular grids. Note also that the frontal method inherently combines assembly and reduction.

Frontal solvers are typically organized into 3 main sections; prefront, front and backsubstitution. The prefront constructs the table of first, intermediate and last appearances illustrated in Table 2.4. The front performs the assembly and reduction, and the backsubstitution the usual solution computation. All 3 sections are generally coded with 2 main nested loops. The outer loop on the elements and the inner loop on the number of DOF for that element. Of course other loops must be nested inside these two main loops to perform the reduction and backsubstitution. It is

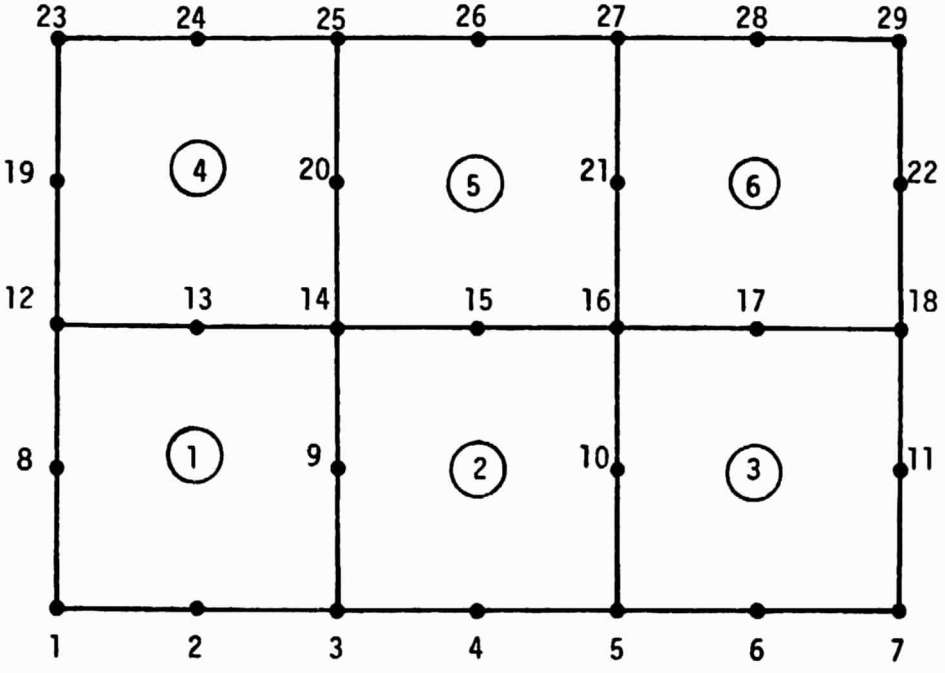


Figure 2.5 Example Grid of Q8 Elements

	Current Element					
Node	1	2	3	4	5	6
1	1*	-1	-1	-1	-1	-1
2	2*	-1	-1			
3	3	3*	-1			
4	0	1*	-1			
5	0	2	2*			
6	0	0	1*			
7	0	0	3*			
8	4*	-1	-1			
9	5	5*	-1			
10	0	4	4*			
11	0	0	5*	-1		
12	6	6	6	6*		
13	7	7	7	7*	-1	
14	8	8	8	8	8*	
15	0	9	9	9	9*	-1
16	0	10	10	10	10	10*
17		0	11	11	11	11*
18			12	12	12	12*
19			0	1*	-1	-1
20				2	2*	-1
21				0	1	1*
22				0	0	2*
23				3*	-1	-1
24				4*	-1	-1
25				5	5*	-1
26				0	3*	-1
27				0	4	4*
28				0	0	5*
29	0	0	0	0	0	6*
max front	8	10	12	12	12(10)	12(8)

Table 2.4 Progressive Frontal Locations for Grid 2.5

possible to code pre-prefronts that order the elements in an optional or sub-optional manner so as to minimize the front. This is similar to band minimizers and was not considered in the present study. Front and band minimizers are not useful for regular grids.

The frontal method is especially efficient in handling grids with midside nodes as shown in Figure 2.5 and Table 2.4 where the maximum front is 12 nodes. The maximum bandwidth would be 14 nodes. In general if there are N elements across the width of a uniform grid of Q8 elements with 2 DOF per node, the front f=2(2N+6) but the band b=2(3N+5). Thus, in the limit as $N\to\infty$ $f/b\to 2/3$. The frontal method is also much more efficient in storage and computations. The frontal method requires storage for $K_f=f^2/2$ coefficients whereas the band method needs $K_b=2b^2$, thus in the limit as $N\to\infty$ $K_f/K_b\to 1/9$, almost an order of magnitude. The number of computations is proportional to f^2 and b^2, respectively, thus the ratio of front to band approaches 4/9 as $N\to\infty$.

The frontal method loses its computational advantage for grids without midside nodes since f=b. However, it still retains a significant storage advantage of 1/4.

2.2.3 Column or Skyline Solvers

Column solvers are currently very popular among the coterie of developers of general purpose finite element codes (e.g., SAP [7,8] and ADINA [9]) and there have been many recent papers in the literature describing them [10,11,12,13]. Column solvers are based on the skyline storage scheme illustrated in Figures 2.6-2.8 in which a nonuniform skyline is drawn covering the minimum number of nonzero coefficients that can be identified in each column. Emphasis is supplied on the word identified because the user generally must perform the identifying and numbering shown in Figures 2.7-2.8 and this may not be a trivial task. Another possibly troublesome feature of these solvers is that they usually do not assemble the element stiffness coefficients, again this nontrivial task is left to the user.

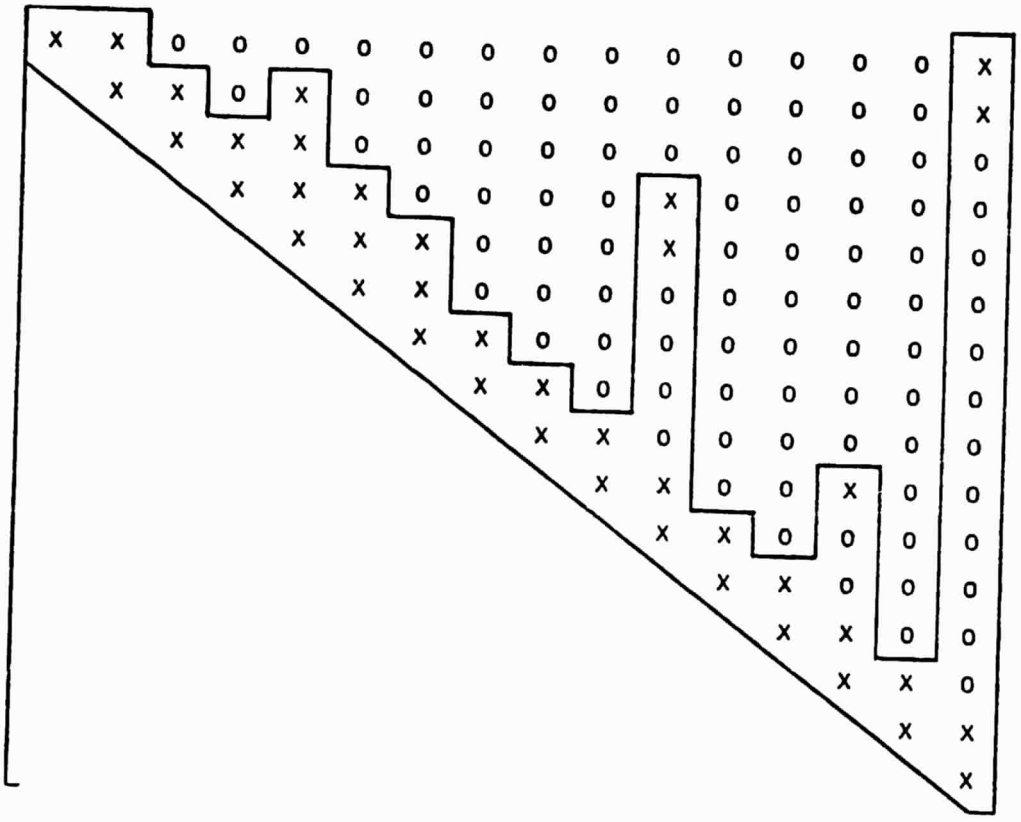
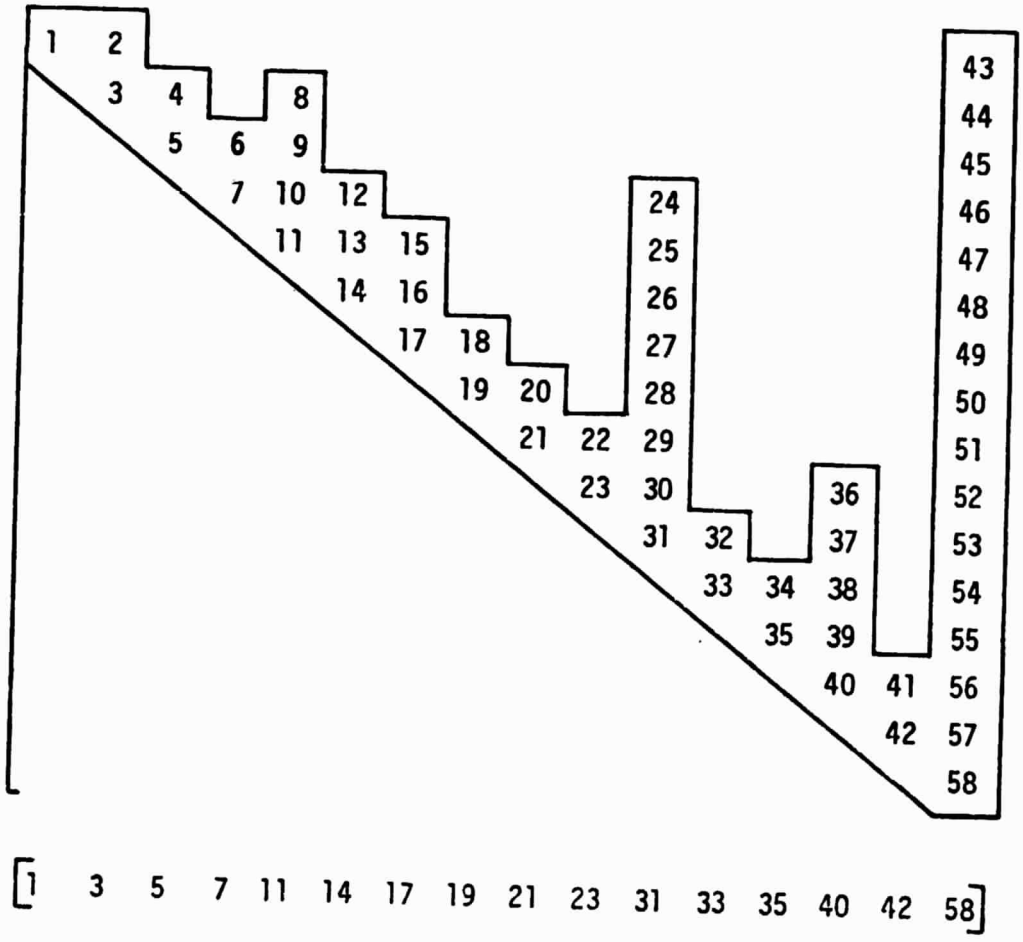


Figure 2.6 Typical Skyline



[1 3 5 7 11 14 17 19 21 23 31 33 35 40 42 58]

Pointer Array of Location of Diagonal Terms

Figure 2.7 Storage Scheme for Column Solver

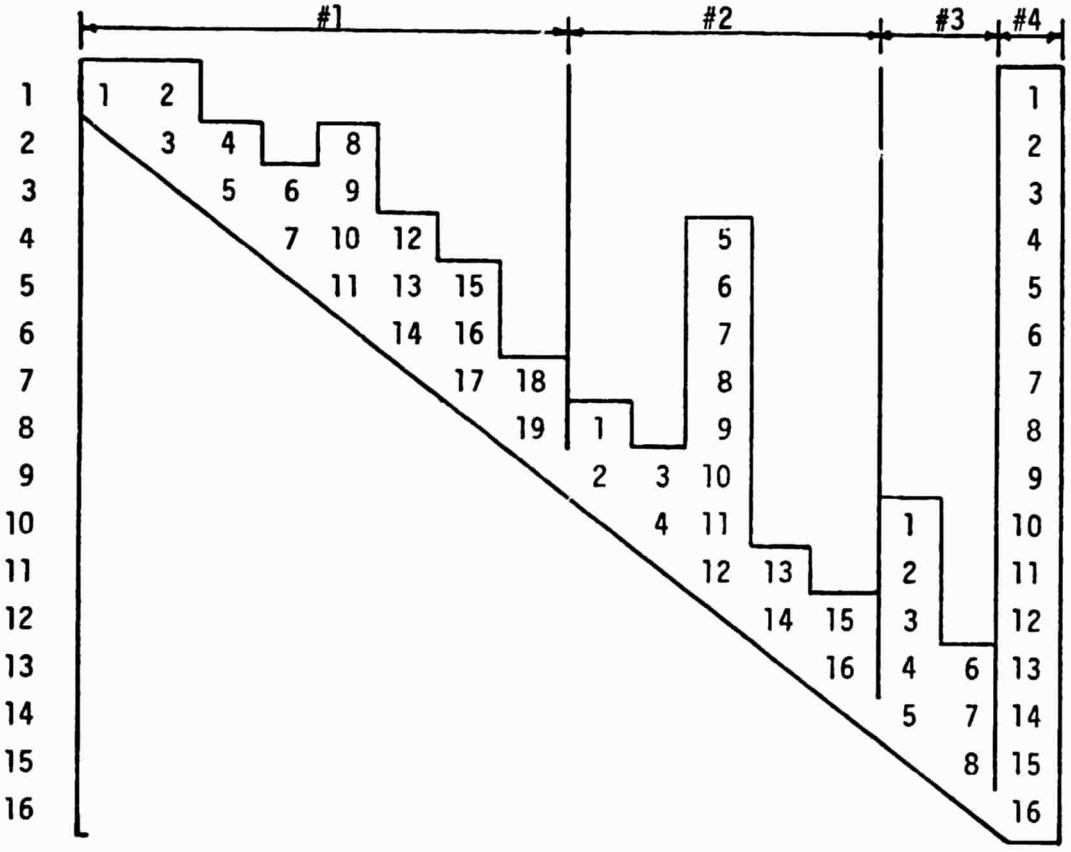


Figure 2.8 Example of Skyline Stored in 4 Blocks

For entirely in-core column solvers the tasks of reduction and back-substitution are straightforward and essentially the same as for band and frontal solvers. However, when the equations are too large to fit into core they must be blocked as illustrated in Figure 2.8. The reduction and backsubstitution phases here are considerably more complicated and unfortunately can require substantial I/O. In the simple illustrated example, the reduction of block 2 requires that only block 1 be available. Similarly, the reduction of block 3 requires that only block 2 be available. But block 4 needs blocks 1, 2 and 3! The minimum working storage for our example would be 2 blocks (about 40 words), and if this was the maximum available core, it would be necessary to write and read blocks 1 and 2 twice. Thus, more than 1 word of I/O is required for each coefficient (e.g., as required in frontal schemes).

Another obvious I/O inefficiency with column solvers is that the assembly and reduction phases are separated, thus requiring at least twice the I/O of frontal solvers. However, there are significant advantages to column solvers. For example, in the equations illustrated in Figures 2.6-2.7 only 58 words are necessary to store the active coefficients. Since the last column is full, frontal and band solvers would need to store the entire upper triangular region of 136 words. The column storage scheme would also require fewer computations for the same reason. This advantage is not effective in the regularly number cartesian grids generally used in 2D or 3D continuum problems and for this reason column solvers may be more effective for special applications with sparse matrices or very general purpose computer codes like SAP [7] and ADINA [9].

To illustrate the comparison between column and frontal solvers consider the grid of Q8 elements shown in Figure 2.9a & b. Both grids have 4 elements across the minimum connectivity direction. The column numberings in Figure 2.9a are like a band solver would number, and Figure 2.9b gives numberings similar to a frontal scheme. Figure 2.10 shows the skyline for the band numbering. In comparing storage in Figure 2.10 to the frontal scheme which would have a virtually constant front f=14, it is

seen that the number of terms in the skyline is virtually the same as the number within the superimposed front. In Figure 2.11 a similar skyline is shown for the frontal numbering scheme of Figure 2.9b. Here there are about 10% fewer terms in the skyline than the front. This potentially could lead to a 20% reduction in computations, but recall that at least twice as much I/O must be done by the column solver.

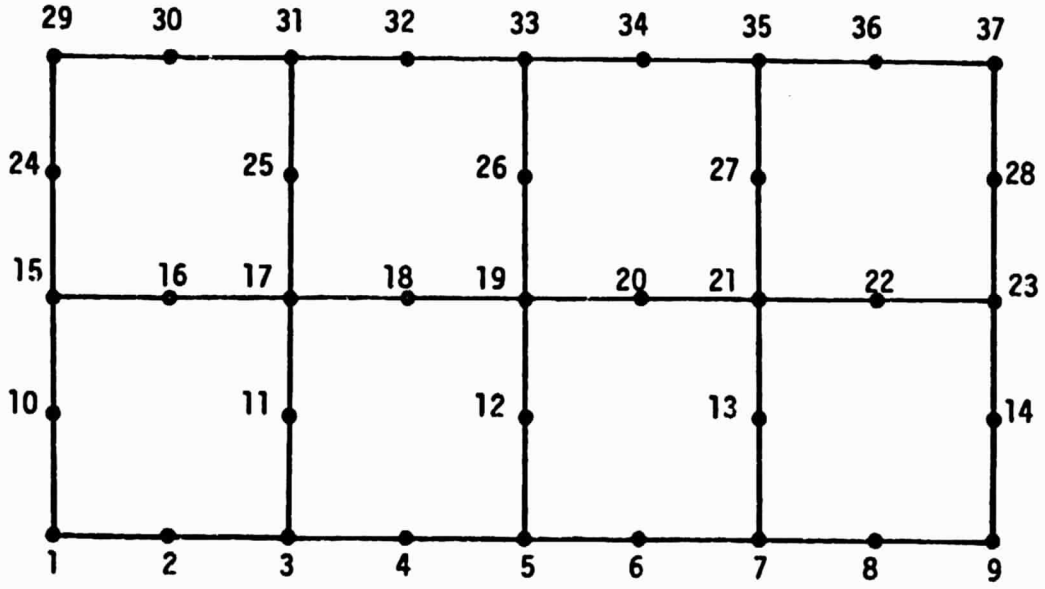


Figure 2.9a Band Oriented Numbering

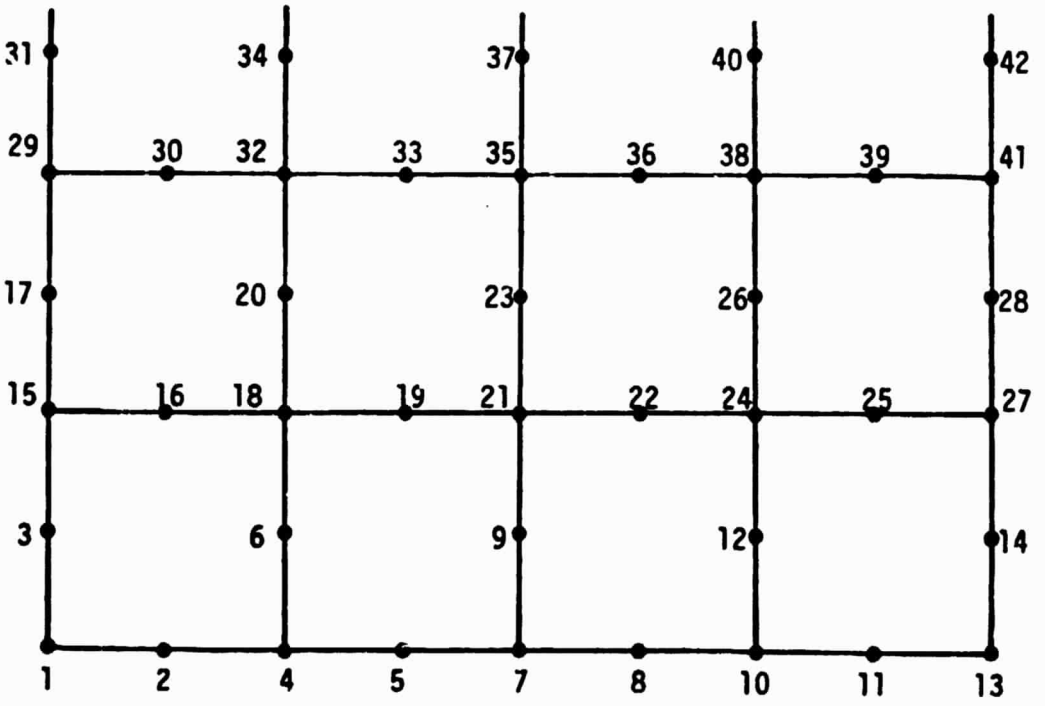


Figure 2.9b Frontal Oriented Numbering

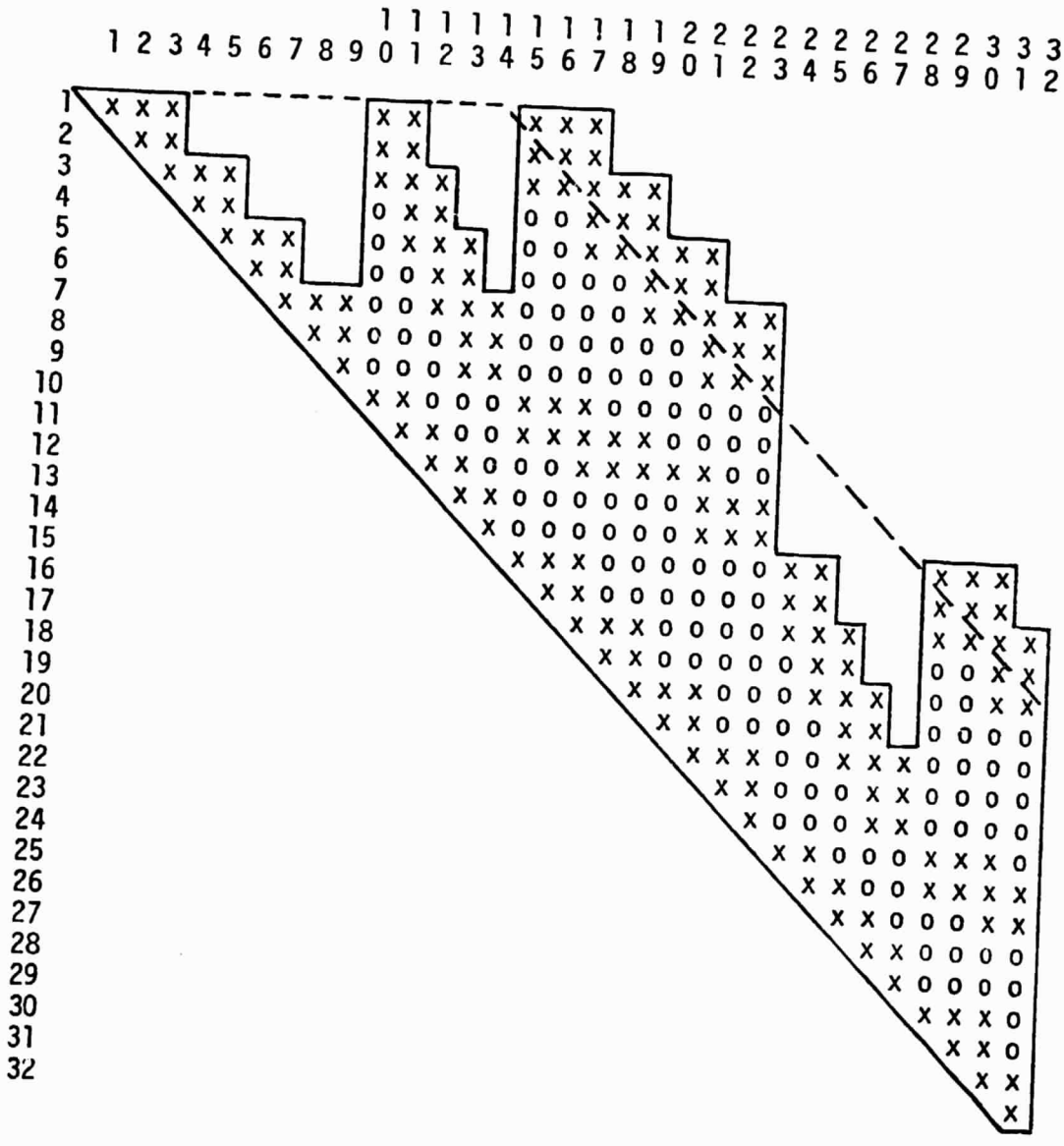


Figure 2.10 Skyline for Band Numbering

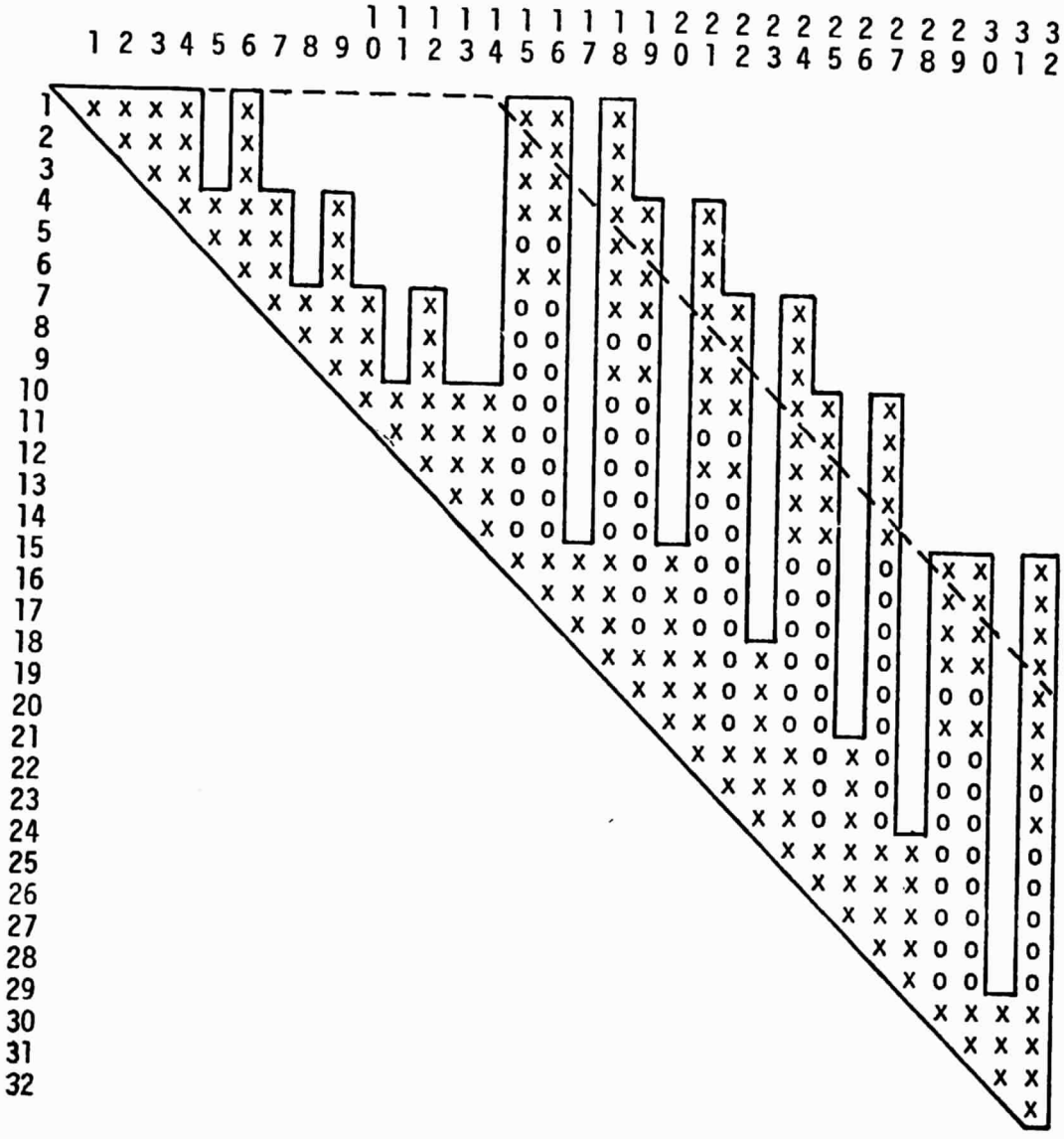


Figure 2.11 Skyline for Frontal Numbering

CHAPTER 3 CHARACTERISTICS OF SPECIFIC SOLVERS EVALUATED

3.0 General Remarks

Due to the scope and funding of the present study, it was not possible to obtain optimally coded solvers of all types. Since virtually all computer systems are unique and require specialized coding to operate at peak efficiency, this means that those imported solvers will be at a disadvantage to those that were coded on site. While it was not possible to extensively recode the imported routines, we did attempt to use these solvers fairly in our study. This does not mean that our final evaluations will be fair or impartial. In fortran programming there is always the not-invented-here syndrome to be contended with.

Also, there was no attempt to make an exhaustive search to locate new and innovative routines. Rather, the author chose to use routines that were generally available and known to him a priori. Clearly, many of the better solvers may be not generally available because they are proprietary or because the developer does not code an easily transferrable modular routine [14].

The following is a brief description of the actual band, frontal and column solvers that were used in the present study.

3.1 Bandsolvers

3.1.1 BANSOL

BANSOL is a general purpose out-of-core equation solver that stores and operates on the coefficients in bandwidth blocks as illustrated in Figure 3.1. BANSOL is a very general standard utility routine that was originally coded by E. L. Wilson and subsequently modified by the author when used in the early finite element codes PALOS [15] and NAOS [16]. It is an out-of-core solver in that the reduction takes place block by block with at most two blocks in core at a time. When the reduction of a block is complete, it is written onto low speed storage and the next block is formed

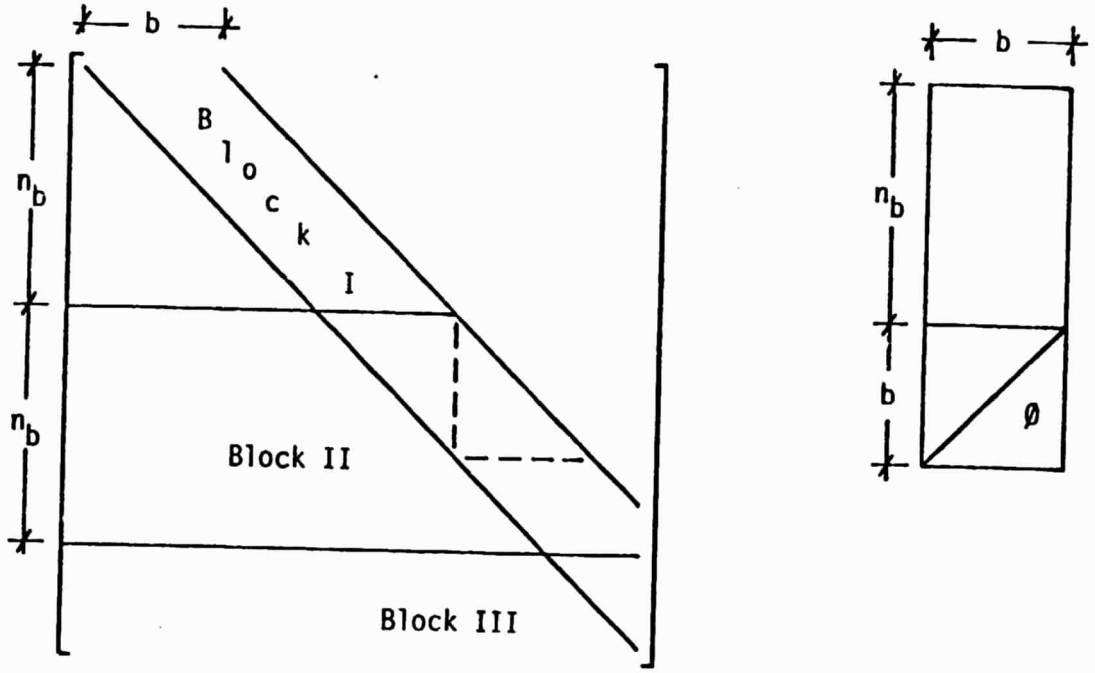


Figure 3.1 Typical BANSOL Storage Scheme

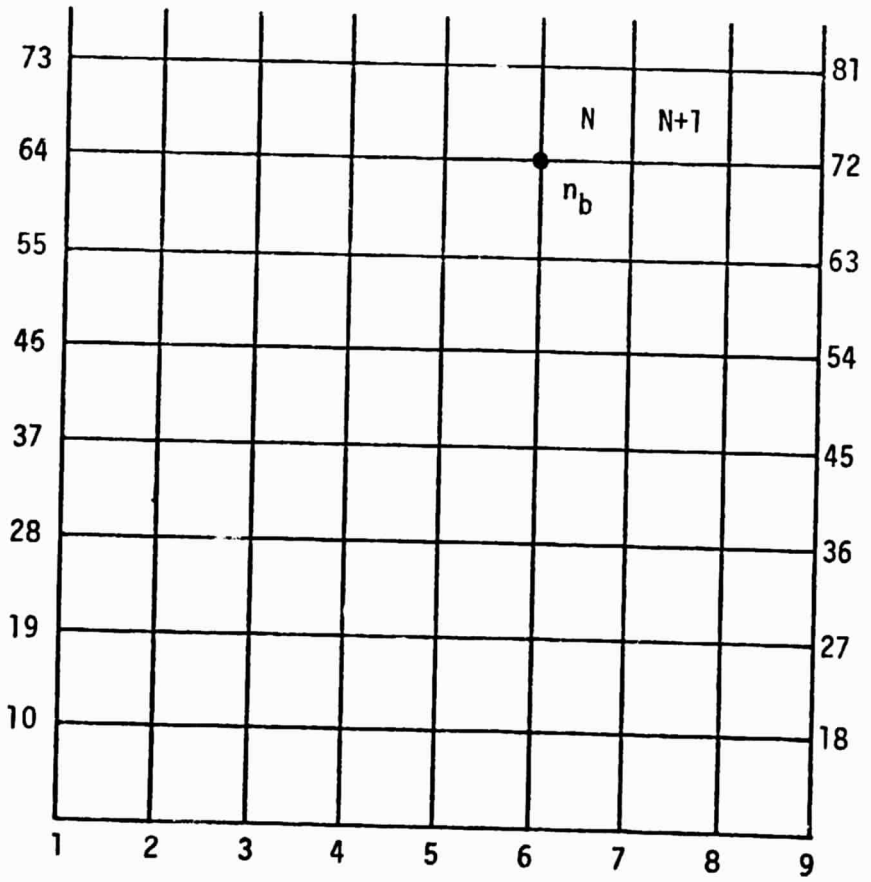


Figure 3.2 Typical Q4 BANSOL Grid

and reduced. The minimum storage required for Bansol is $2b^2$. If more storage is available the number of DOF that can be stored per block is increased according to the formula n_b = NDIM/b - b where NDIM is the available core storage and b is the bandwidth.

For the grid shown in Figure 3.2, after element N is assembled, the equations up to point n_b are complete. The coefficients that have been assembled up to element N are stored as shown in Figure 3.1 in two-dimensional array form using singlely subscripted arithmetic. Forward elimination for equations 1 to n_b is done and then the revised coefficients from equations 1 to n_b are written onto low speed disk storage and the remaining triangular portion is shifted upward so that equation n_b+1 in block 1 becomes equation 1 in block 2. The process is then continued as before assembling the stiffness matrix from the N+1 element. Note that BANSOL combines assembly and elimination and consequently avoids the extra I/O charges for the additional write and read required when these phases are done separately.

BANSOL uses an efficient high speed binary I/O routine and does not use standard fortran I/O. However, since the usual block size is 10^4 words or larger, this is not an important factor.

BANSOL is generally useful for solving small to intermediate sized systems of equations associated with 2D elements without midside nodes ($b \leq 150$). When $2b^2$ exceeds the available core requirement, other solvers must be used.

BANSOL was used in the study as a base line comparison for the intermediate sized grids and of course is not effective for large 2D or 3D problems with larger bandwidths.

3.1.2 USOL

USOL is an out-of-core band solver that has the capability of partitioning the band when the bandwidth blocks exceed the available core capacity. USOL was developed by E. L. Wilson and was the original solver used in the

SAP codes [7]. USOL performs the eliminations in essentially the same manner as BANSOL, however, the process of assembly and reduction are separate. No special coding was done on USOL to optimize its logic or I/O efficiency, since it was anticipated at the outset of this research that bandsolvers that partitioned the band would not be competitive. Also, it is generally known that USOL is not an efficient solver for grids with midside nodes or for large bands. It was used in SAP because of its ability to use available core.

3.2 Frontal Solvers

3.2.1 ZIPP

ZIPP is the very general frontal solver developed by Irons [17] and used extensively in the TEXGAP computer programs [1]. In this report there are 3 versions of ZIPP that are used. The first is the original deck as coded by Irons and listed in his paper [17]. The original version used Fortran I/O and was written to include many options for multiple load vectors and resolutions for nonlinear problems. The first modification converted all I/O and removed some burdensome logic related to the multiple load vectors and resolutions. This version is called HSZIPP1 for High Speed binary I/O. Since ZIPP uses its own dimensioned buffer [17] there is greater control possible over when the internal buffer is released to the I/O buffer (or transferred directly to disk). Irons' original version released the dimensioned buffer after each element had been processed, thus the typical number of words transferred was 500-2000. A simple modification was made so that the internal dimensioned buffer was dumped only when it was full, thus 5000 words are usually transferred. This second version is called HSZIPP2.

As described in Section 2.2.2, ZIPP permits a very general node labeling scheme to be used because it has an extensive prefront routine that stores all DOF labels and then tests all other labels to determine the necessary information about the first, intermediate and last appearances of a label. This is a computationally expensive procedure that increases as N^2 where N equals the number of degrees-of-freedom per element times

the number of elements (N≈4n for 2D problems). This particular prefront feature is not common to all frontal solvers, for example, PUZZLE does not perform this extensive search.

3.2.2 PUZZLE

PUZZLE is a frontal solver developed by C.P. Johnson which can partition the front when the front size $f^2/2$ exceeds the available core [18]. Another important feature of this code is its extensive substructuring capability. The frontal method intrinsicly generates substructure stiffnesses as the front progress, and the coding in PUZZLE fully exploits this useful feature. PUZZLE operates in a similar manner to ZIPP although the coding is quite different. The PUZZLE prefront is less general than ZIPP and is considerably more efficient. For typical continuum problems this loss in generality is minor.

Like the ZIPP versions, PUZZLE handles all I/O using the high speed IOP routine and operates at near peak I/O efficiency on the Texas system. A very significant portion of this I/O efficiency occurs with large fronts that require partitioning of the front. Say for example that the front is almost constant and core memory is available only for about 1/2 of the needed space. Thus, 2 partitions are necessary for all frontal blocks. With USOL, COLSOL and most other band partitioning or column solvers this would at least twice as much I/O on the coefficient scratch file. However, PUZZLE is coded in such a way that needless repetition of I/O is minimized and frequently reduces the I/O burden to 1.3-1.5 times that required if all coefficients fit in-core. For problems that require 3-10 frontal partitions, this savings could be 1-2 orders of magnitude.

3.3 Column Solvers

3.3.1 SKYSOL

SKYSOL is an in-core skyline or column solver developed by C. A. Felippa [12]. Since SKYSOL is in-core, the maximum capacity is limited

to problems with 400 to 1000 DOF. In situations where there are only a few long columns, it is especially efficient in reducing storage as illustrated in Figure 2.6. SKYSOL is useful for one-dimensional and some small two-dimensional problems. It is used in this research to provide "base-line" comparisons, since it is anticipated that all out-of-core solvers will be less efficient.

The method used for storing the coefficients is the compacted column storage scheme discussed in Section 2.3.2. All the elements above the main diagonal are stored in sequence by columns. All coefficients above the first nonzero coefficient in each column are omitted and all coefficients between that point and the main diagonal are stored whether or not they are zeros. To use the compacted storage scheme, a locator array is needed to locate the position of the diagonal coefficients in the one-dimensional array. The locator array gives the necessary information to assemble the total stiffness and determines how many elements need to be revised during the elimination procedure of a given coefficient.

3.3.2 GASP

GASP is an active column or skyline solver coded by C. P. Johnson at the University of Texas at Austin primarily for the analysis of thin shell structures (which usually have 5 or 6 DOF per node). Since GASP was not intended for use in a general purpose code, the column heights are restricted to be such that they do not extend past the previous block. For example, in Figure 2.8 the column heights in block #3 extend only part way into block #2 but block #4 extneds back to block #1. In GASP, block #4 would be restricted to a height of 8. This restriction is important in limiting I/O since multiple passes through the stiffness coefficient scratch file are eliminated.

GASP was coded using the high speed binary I/O routines available at Texas, thus further enhancing its I/O efficiency. As with all column solvers, GASP must separate the assembly and reduction phases and make multiple passes

through the element stiffness file. However, cumulative lower and upper limits on these passes are computed to reduce needless I/O. GASP requires the nodal connectivity array to be stored in-core and requires that the array contain DOF numbers for purposes of storage (gaps are permitted).

3.3.3 COLSOL

COLSOL is an active column solver coded by Wilson and Dovey at the University of California to replace USOL in the SAP codes [8]. Like USOL, COLSOL can handle virtually unlimited bandwidths or column heights or conversely can fit into very small core when necessary. Thus, COLSOL must handle column heights that extend past the previous block, e.g., block #4 in Figure 2.8. All I/O in COLSOL is unformatted fortran I/O and because COLSOL was obtained late in this study it was not possible to improve it. For this reason, timings on COLSOL are likely to be slower than for more optimized coding.

CHAPTER 4 COMPARISONS BETWEEN THE VARIOUS SOLVERS

4.1 Basis of Comparison

The main thrust of this research is to compare the computational efficiency of the various methods of solving systems of equations by performing controlled numerical experiments on the solvers described in Chapter 3. The basis of comparison will be the "computer time" required by each solver to effect the solution of a set of model problems.

It is extremely important in evaluating this particular research to at all times remember the nature of the reported computer timings. All results presented herein were obtained on the University of Texas 6600/6400 operating system. This system permits two types of timing; a "central processor(CP) clock" and a charge time (TM). TM time equals CP time (sec) + 0.004 PRUs where 1 PRU = 64 (60 bits) words read or written. All times are in seconds. Every possible effort was made to run the model problems on the 6600 because the 6400 gives "equivalent 6600" time. In practice, it is found that runs on the 6400 were generally 10-50% longer than on the 6600. In spite of our efforts to eliminate this variable, it is possible that some of the times reported are from 6400 runs. In any event, CDC 6600s are well known for their poor "clocks" and variations of $\pm$15% are possible depending on how many "rollouts" of the job are made. Generally, jobs are run during peak demand periods give longer times than those run at low demand periods. In spite of all these difficulties, it was still possible to adequately interpret and compare results and frequently duplicate runs gave variations as low as $\pm$1%.

When comparing equation solvers it is important to establish a common basis of comparison, or the results of the experiments could be meaningless. To this end, all equation solvers were required to read a sequential file of element stiffness coefficients (including the right hand side), assemble the coefficients, reduce the coefficients (forward elimination) and backsubstitute to compute and print the solution. The actual element stiffness coefficients used were:

$$A_{ii} = 1.0, \quad i=1,2,....N$$

$$A_{ij} = 10^{-20}, \quad i \neq j=1,2,....N$$

$$B_i = 1.0, \quad i=1,2,....N$$

This permitted a check on the solution since $x_j=1$ to machine accuracy (14 figures) and avoided the special case of $A_{ij}=0$ for $i \neq j$ which could lead to erroneous comparisons since some solvers skip on zeros. The sequential file of stiffness coefficients was generally written and read using standard FORTRAN unformatted I/O so that a minimum charge of 512 words (8 PRUs) was incurred for each read or write. Some of the runs (especially the ZIPP runs) were made using the high speed binary I/O routine IOP and this reduced these I/O changes by up to a factor of 10. However, since most of the problems required only one pass through the element tape this effect was negligible since most of the I/O takes place when the assembled coefficients are read and written.

Equation solvers also need information on the connectivity of the elements and in general this information was also passed to the solver by means of a sequential file. However, it was not possible to demand a like effort from each solver because of the particular coding in the solvers that were available. For example, ZIPP reads such a file and takes the information in the file as labels from which a frontal storage position is computed. PUZZLE performs a similar function but requires the labels to be integer numbers from 1 to N. Gaps are permitted, but storage is allocated for N numbers; a much more restrictive form than ZIPP which creates the numbering. USOL and COLSOL both require degree-of-freedom numbering a priori so that no bookkeeping need be done in the solver. Clearly, no meaningful comparisons can be established for the connectivity-bookkeeping phase. However, this is not a troublesome point since in all solvers except ZIPP this time is small. ZIPP does require a substantial amount of time in its prefront and for that reason it is reported separately.

Whenever possible, all identifiable times have been reported; these include:

(1) prefront and/or bookkeeping
(2) assembly
(3) forward elimination
(4) backsubstitution
(5) total time in solver

Because of their intrinsic nature, it is not possible to provide separate timings on each phase for each solver. For example, ZIPP and BANSOL combine assembly and forward elimination, but USOL and COLSOL separate them at the expense of an extra write and read of the entire system of equations.

In spite of the aforementioned difficulties, it is still possible to make meaningful comparisons between the solvers by simply monitoring the total CP and TM time required for each solver.

4.2 Comparison Problems

The most important item in the comparison basis is of course the set of sample problems. This research was directed at large out-of-core solvers and for that reason intermediate and large two-dimensional (2 DOF per node) and small three-dimensional (3 DOF per node) problems were chosen. It was not necessary to select intermediate or large 3D problems as the results will show.

At the present time the majority of finite element production problems still seem to be solved with first order or linear elements, that is, elements with only corner nodes, although use of the quadratic elements (elements with 1 midside node per edge) is increasing. For that reason, and because this research was partially aimed at the TEXGAP computer codes, the following element types were used:

Q4-4 node quadrilaterals (8 DOF)

Q8-8 node quadrilaterals (16 DOF)

B8-8 node bricks (24 DOF)

B20-20 node bricks (60 DOF)

All of the model problems were uniform rectangular grids in 2D and cubic grids in 3D. These are highly specialized grids but they are probably representative of the majority of production problems. In any case, it is doubtful that grid regularity affected the results appreciably. Unless there is some particular feature of the equations (e.g., extremely sparse fronts or bands) that can be readily identified and taken advantage of in the solver, the results for irregular grids will be similar. Of particular importance in the choice of model problems is the presence or absence of midside nodes. If there are only corner nodes in the grid (Q4 or B8 elements) then the front and band are equal. However, if midside nodes are present the front is always less than the band and frequently significantly less. Midside nodes are very troublesome to bandsolvers and run the band up considerably, thereby increasing storage and solution time by approximately the square of the band. Midside nodes cause no difficulty with frontal or column solvers both of which appear to treat these nodes efficiently with frontal solvers perhaps having a small edge. Frontal solvers generally perform less efficiently on grids using only corner nodes since only a few DOF per element are eliminated. In general, band, frontal and column solvers do about the same computations for grids with no midside nodes so the solvers with the least bookkeeping will always perform best. Bookkeeping is generally smallest in bandsolvers and highest in frontal solvers.

The first model experiments were run on the grids shown in Figures 4.1 and 4.2. In Figure 4.1 is shown a grid of 10 x 24 = 240 Q4 elements and 11 x 25 = 275 nodes with 2 DOF per node for a total of n=550 DOF. Test Problem #1 was run on this grid with the numbering across the 10 element direction so that the band(b) and front(f) widths were equal,

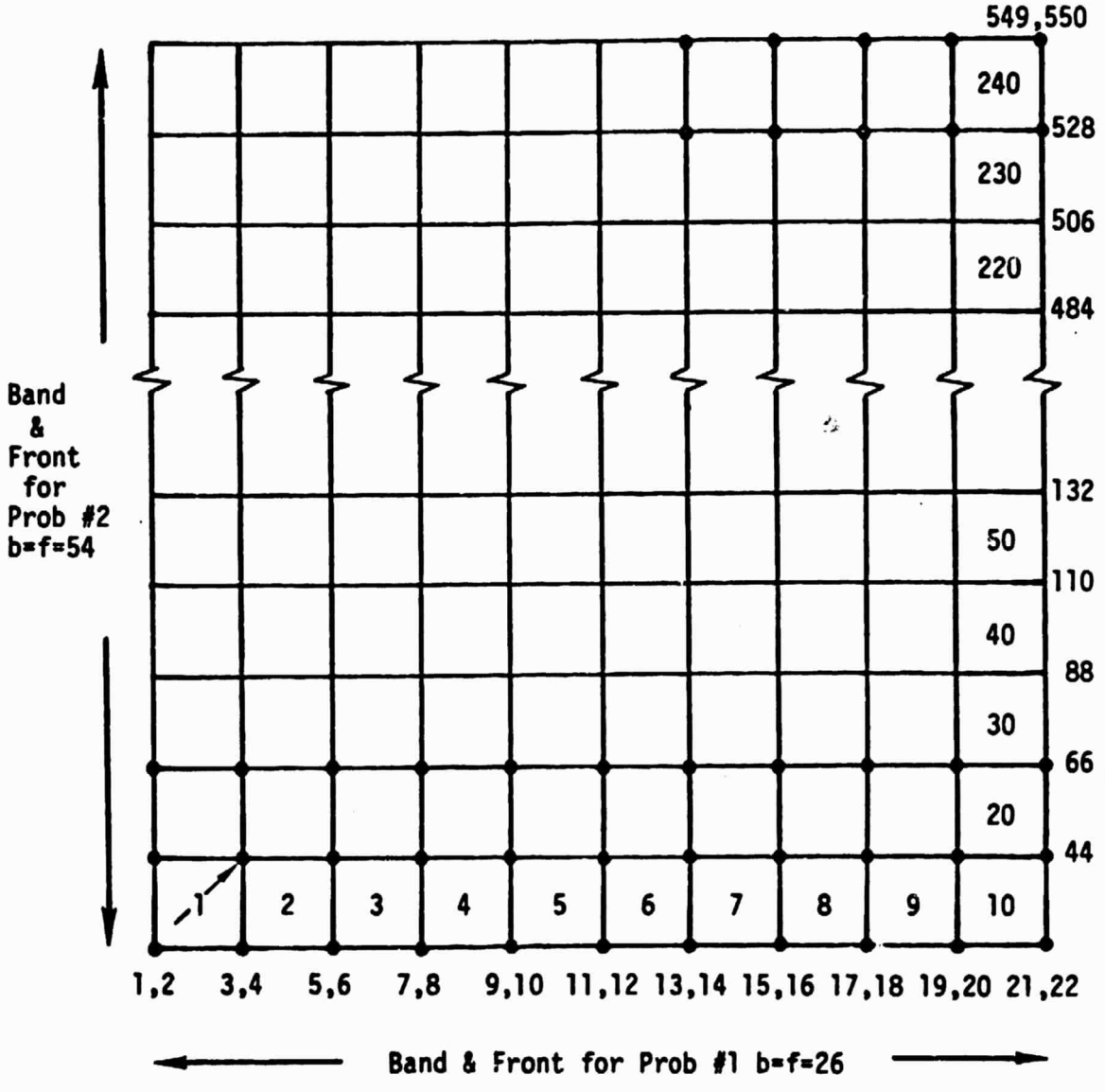


Figure 4.1 Test Problems #1 & 2 for the Q4

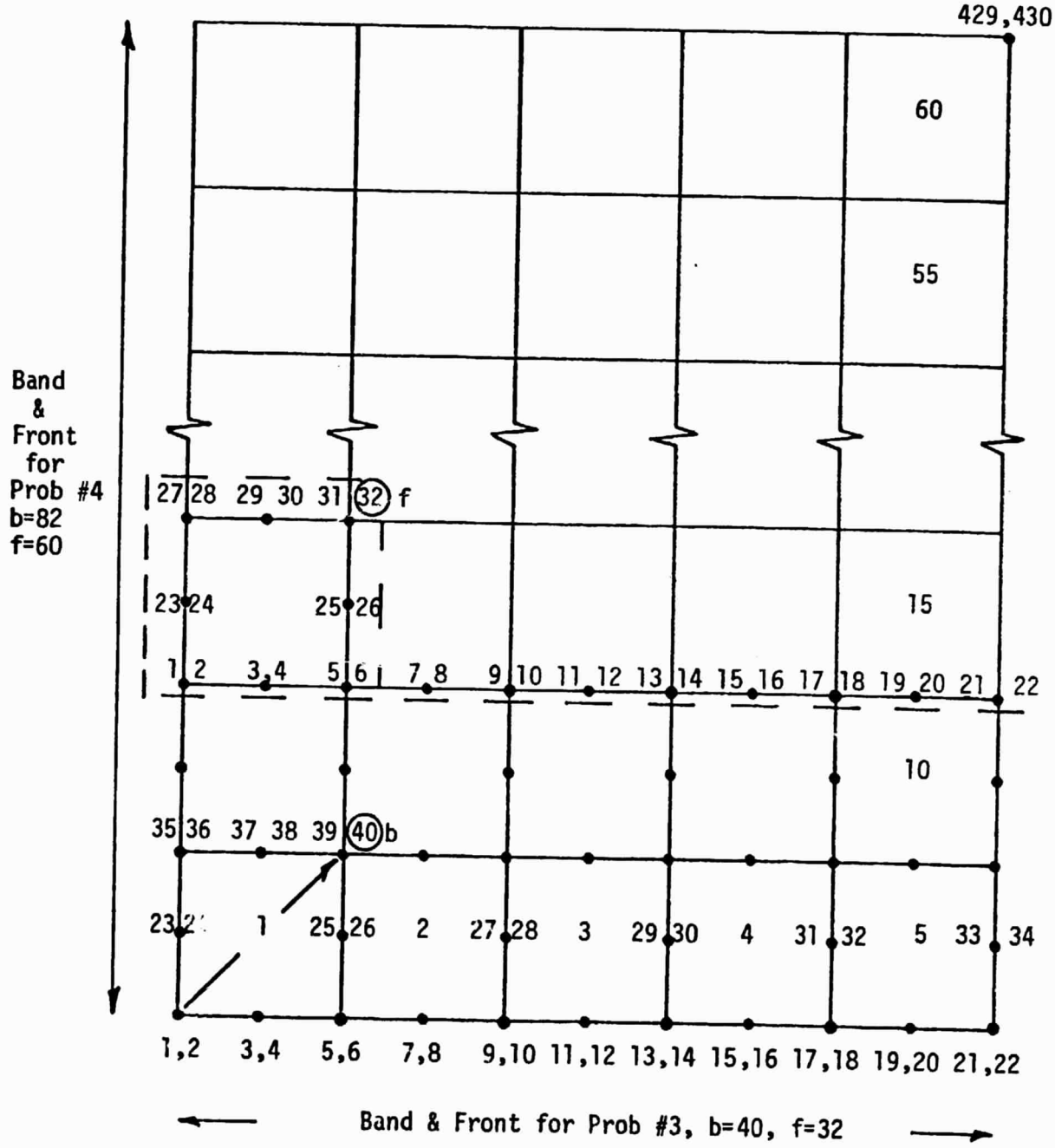


Figure 4.2 Test Problems #3 & 4 for the Q8

b=f=26. Test Problem #2 was run on the same grid but numbering across the 24 element direction so that b=f=54. In Figure 4.2 is shown a grid of 5 x 12 = 60 Q8 elements and 11 x 25 - 60 = 215 nodes and 430 DOF. Test Problem #3 was run on this grid with the numbering across the 5 element direction so that b=40 and f=32. Test Problem #4 numbers across the 12 element direction so that b=82 and f=60.

These four problems are not large but neither are they small in size and they were selected because the general resolution between the Q4 and Q8 grids will be comparable. Thus, for a given modeling the four problems give a range of bands and fronts while retaining approximately the same number of DOF. The missing 60 nodes in problems 3 & 4 could be handled by a frontal or column solver with little additional computational effort.

4.3 Results

The timings for problems 1-4 are given in Tables 4.1-4. Similar results are also given in Tables 4.5-6 for a 36 x 10 grid of Q4, and an 18 x 5 grid of Q8s. This had the effect of increasing the band and front; n=814, b=f=78 for Test Problem #5 and n=634, b=118, f=84 for Test Problem #6.

The results reveal that as expected the in-core SKYSOL is the most efficient in both total CP & TM time for the solution. When considering TM time, BANSOL was always the next most efficient and usually was close on CP times. Note that BANSOL & USOL were faster for CP times than the frontal and column solver for problems 1, 2 & 5, the Q4 problems, but were slower for problems 3, 4 & 6, the Q8 problems. This is caused by the midside nodes running the bandwidth up and demonstrates the general superiority of frontal and column solvers for grids with midside nodes.

The CP times reported for USOL are generally comparable but the TM times are very poor. This is caused by high I/O changes from unnecessary I/O such as forming the equation separately before beginning the reduction process. This doubles the I/O changes because if sufficient core is available to store the full band, front or active columns and no secondary partitioning is necessary, then each stiffness coefficient must be written and

	SKYSOL	BANSOL	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	3.26	6.94	25.27	61.08	40.98	25.56	26.91		
CP(sec)	2.22	3.05	4.73	7.26	5.70	5.08	6.09		
Working Array Storage	13,441	12,000	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	3.54	6.83	29.34	78.20	38.54	23.72	35.38		

Table 4.1 Test Problem #1 10x24 Grid of Q4's
n=550 b=f=26

	SKYSOL	BANSOL	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	5.56	15.17	37.77	64.79	46.52	29.60	33.01		
CP(sec)	4.53	6.62	4.36	11.00	10.25	8.01	10.50		
Working Array Storage	26,881	12,000	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	5.29	10.48	42.19	78.20	39.50	24.84	28.16		

Table 4.2 Test Problem #2 24x10 Grid of Q4's
n=550 b=f=54

	SKYSOL	BANSOL	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	3.28	7.29	13.81	17.75	12.98	9.46	9.97		
CP(sec)	2.22	3.70	3.95	4.21	3.26	3.13	3.08		
Working Array Storage	13,329	12,000	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	3.66	6.46	12.88	20.57	11.77	8.35	13.53		

Table 4.3 Test Problem #3 5x12 Grid of Q8's
n=430 b=40 f=32

	SKYSOL	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	5.66	34.00	20.37	17.37	12.43	13.01		
CP(sec)	4.63	8.18	6.54	6.97	5.04	5.45		
Working Array Storage	25,929	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	5.52	28.83	20.86	12.40	9.44	12.03		

Table 4.4 Test Problem #4 12x5 Grid of Q8's
n=430 b=82 f=60

	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	97.9	103.4	75.9	54.4	60.4		
CP(sec)	14.8	22.8	19.8	18.1	24.4		
Working Array Storage	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	95.7	116.6	60.5	40.7	44.8		

Table 4.5 Test Problem #5 36x10 Grid of Q4's
n=814 b=f=78

	USOL	ZIPP	HSZIPP1	HSZIPP2	PUZZLE	GASP	COLSOL
TM(sec)	83.97	36.06	28.92	25.47	24.99		
CP(sec)	20.41	14.02	12.45	11.92	12.41		
Working Array Storage	12,000	12,000	12,000	12,000	12,000		
I/O PRU's TM(sec) @.004 sec/PRU	67.53	31.99	19.02	18.51	18.40		

Table 4.6 Test Problem #6 18x5 Grid of Q8's
n=634 b=118 f=84

read only once. For this reason, no further studies on USOL were made since it is clearly inefficient.

There are 3 columns of results given for the ZIPP solvers that reflect only improvements in the I/O handling and no computational changes. Note that reductions of up to 60% were made in the TM time and that this also resulted in CP reductions of up to 30%. Inefficient I/O is clearly to be avoided both to reduce wasted (I/O) charges and to increase CP efficiency. The results from HSZIPP2 and PUZZLE are seen to be similar as would be expected since both are frontal solvers and both use efficient high speed binary I/O.

Another important variable reported in Tables 4.1-6 is the storage required for the solver with a nominal lower level cutoff of 12,000 words being taken. For Problem #1, SKYSOL needed only 13,441 words to store all coefficients entirely in core, but BANSOL or USOL would need n*b=14,300 and since only 12,000 were made available they needed at least 2 blocks of equations to effect the solution. In Problem #2 almost twice the amount of core was needed by SKYSOL. BANSOL and USOL needed over 5,000 words just for their $2b^2$ blocks. ZIPP and PUZZLE actually needed only $f^2/2 \simeq 1300$ for their active storage. In Problem #4 SKYSOL needed 26,000 words; BANSOL and USOL needed $2b^2 \simeq 15500$ just for the blocks, but ZIPP, PUZZLE and COLSOL could get by with only 3900 words. Problem #6 gets quite large and would have required $2b^2 \simeq 28{,}000$ words for 2 full blocks and thus with only 12,000 available USOL had to effect a secondary partition - doubling its PRUs. Note that for Problem #6 USOL require over 3 times the TM time and 70% more CP time than ZIPP. ZIPP's storage requirements were still under 4,000 words, a factor of 4 less than needed by USOL to stay in-core.

4.4 Further Results for Frontal Solvers

4.4.0 General Remarks

The conclusion that can be drawn from the results given in Tables 4.1-6 is that HSZIPP2 and PUZZLE consistently give faster CP and TM timings. For problems which do not require subdivision of the front there is little difference between HSZIPP2 and PUZZLE except for the more extensive prefront

used in the ZIPP codes. Because the frontal codes were found to be the more efficient, it was decided to study HSZIPP2 in more detail. Frontal solvers consist of 3 major parts; prefront, front and backsubstitution. In the remainder of this chapter a study will be made of these parts.

4.4.1 ZIPP Prefront

The frontal technique in the ZIPP codes must perform extensive prefront sorting on the connectivity to obtain equation numbers in the front. In so doing, it is necessary to sort through the connectivities element by element as shown by the symbolic Fortran coding in Table 4.7. For this reason, the CP times in the prefront should be uniquely determined by NIZZ. NIZZ = (No. DOF/element)*(No. of element).

```
      NLAST=0
      DO 300 NEL=1,NUMEL
      . . . . .
      DO 200 KL=1,KUREL
      NLAST=NLAST+1
      DO 100 N=NLAST+1,NIZZ
      IF (NIZ(NLAST).NE.NIX(N)) GO TO 100
      . . . . .
  100 CONTINUE
  200 CONTINUE
      . . . . .
  300 CONTINUE
```

Table 4.7 Symbolic Fortran in ZIPP Prefront

Since the sort is triangular the dependence should be proportional to $NIZZ^2$. This is verified by the prefront timings shown in Figure 4.3 for the Q8 elements. Although the logic given in Table 4.7 has some nominal dependence on KUREL and NELEM, the number of degrees-of-freedom per element and the total number of elements, respectively, the strong dependence is on NIZZ, the sum of all degrees-of-freedom in all elements. Thus, the CP times shown

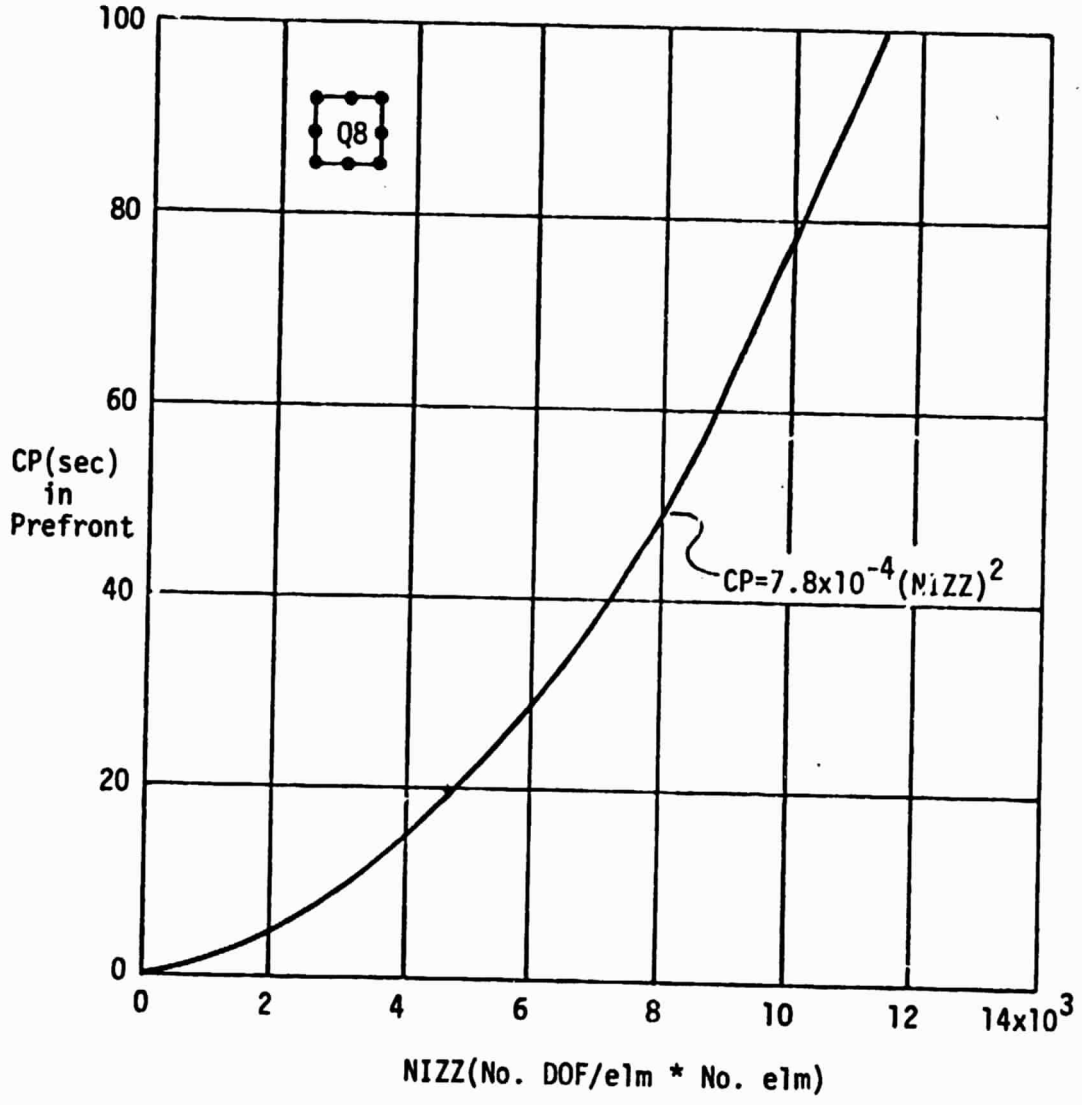


Figure 4.3 ZIPP Prefront Times for Q8

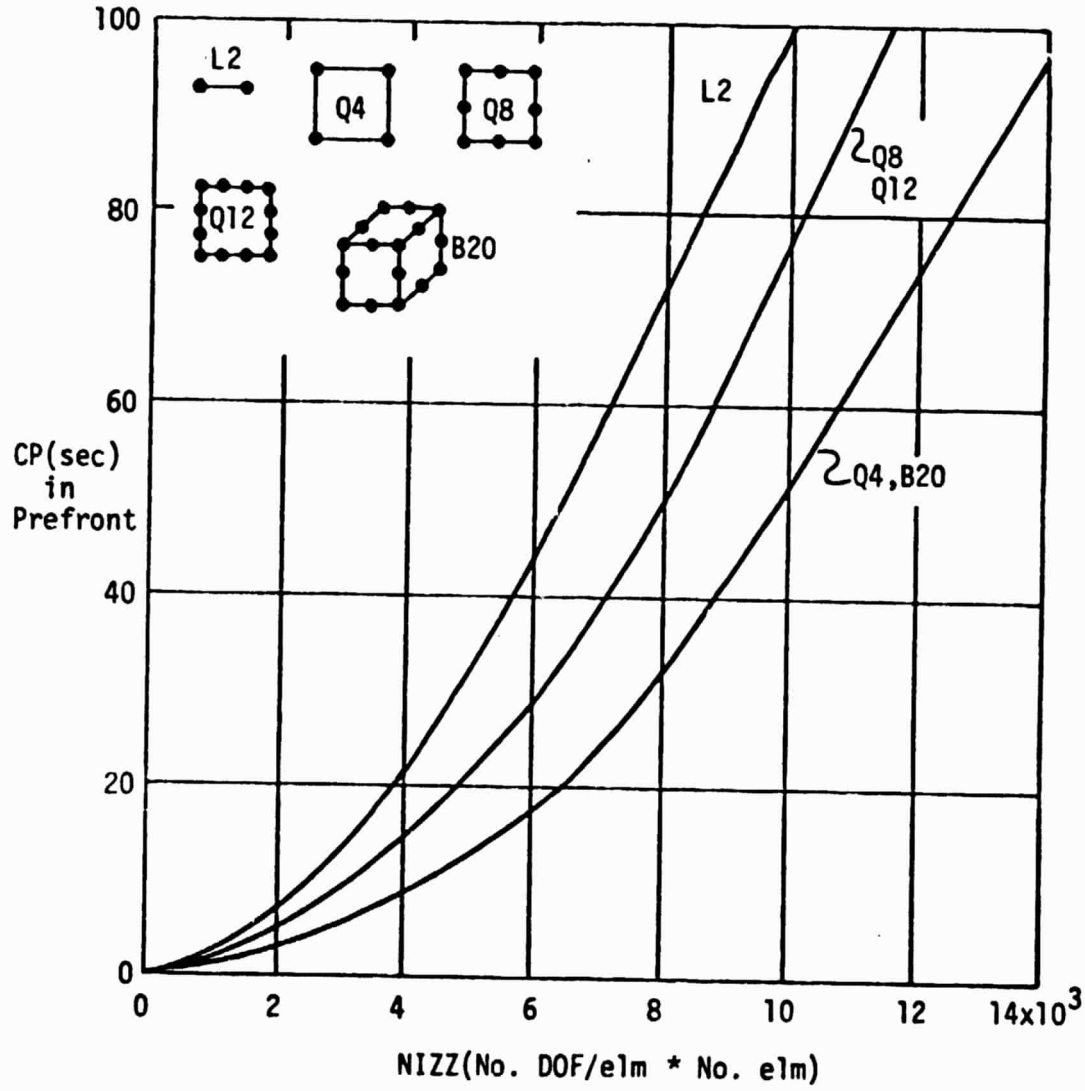


Figure 4.4 ZIPP Prefront Times for Various Elements

in Figure 4.4 for the Q4, Q12, B20 and L2 element types were not expected. Table 4.8 gives the relevant statistics for these element types (the "L" type is one-dimensional, "Q" types are two-dimensional and the "B" type is three-dimensional).

Element	No. of Nodes	No. of DOF per node	Total No. of DOF per element KUREL
L2	2	1	2
Q4	4	2	8
Q8	8	2	16
Q12	12	2	24
B20	20	3	60

Table 4.8 Prefront Test Element Types

As shown in Figure 4.4, for a fixed value of NIZZ, the L2 requires the most time, the Q8 and Q12 less time and the Q4 and B20 the least time. These differences are substantial even for NIZZ as low as 4,000. There is no obvious explanation for these unusual groupings or the strong time differentials.

The prefront times in ZIPP can be significant, but as discussed earlier, this procedure is very general and may not be necessary for many problems. For example, for the B20 element with 3 degrees-of-freedom per node, it is necessary to do the sorting only on the 20 nodes rather than the 60 total degrees-of-freedom in the element. The node scheme will typically be 9 times more efficient.

4.4.2 ZIPP Front

Naturally, the majority of execution is spent in the "front" portion of frontal solvers which consists of both the assembly and reduction phases. Front times have been measured for the Q4, Q8, Q12 and B20 elements.

In Figure 4.5 is plotted CP time for the front portion vs. the total number of equations(n) for various front widths (f=60, 84,112 & 152) using

the Q8 element type. As would be expected, the time varies linearly with n for fixed f. Figure 4.6 is a similar plot for the Q4 element type and reveals similar results. In Figure 4.7 is given a master plot of CP time per 1,000 equations(n) vs. the front width(f) for fronts from 25 to 200 for the Q4, Q8, Q12 and B20 element types. Over this range in front widths, the curve is seen to mildly quadratic, but obviously for front widths between 50 and 200 the curve is nearly linear. This is further illustrated in Figure 4.8 where CP time vs. n is plotted. This indicates that ZIPP handles large fronts efficiently, perhaps improving in efficiency as the front grows (of course, the front must remain in core). It would be interesting to test further in the range of fronts from 200 to 500 (125,000 words of storage) to determine if this weak curvature continues.

4.4.3 ZIPP Backsubstitution

In frontal codes backsubstitution occurs element by element, and as in the prefront, backsubstitution times should be uniquely determined by NIZZ, with the variation nearly linearly. This is verified in Figure 4.9 for all element types. Note that the vertical scale in Figure 4.9 reaches only 12 CP sec for NIZZ≈14,000. This is substantially less than prefront times and can clearly be neglected.

4.4.4 Further PUZZLE Results

The results reported for PUZZLE show that it operates at an efficiency comparable to ZIPP for all of the test problems. Because PUZZLE is the only known frontal code to partition the front, further numerical studies were performed on a larger class test problem. A sequence of tests using the 4x4x2 and 4x4x3 grids of 20 node brick elements (B20, with 3 DOF per node) shown in Figure 4.10 were performed. The reason that 4x4x2 and 4x4x3 grids were run was to determine the incremental time to solve a layer of 4x4 B20 elements. The front for these problems when numbering across the 4x4 plane is 86 nodes or 258 DOF, thus the storage required for the entire front is 33411 words. The results for these runs are given in Table 4.9.

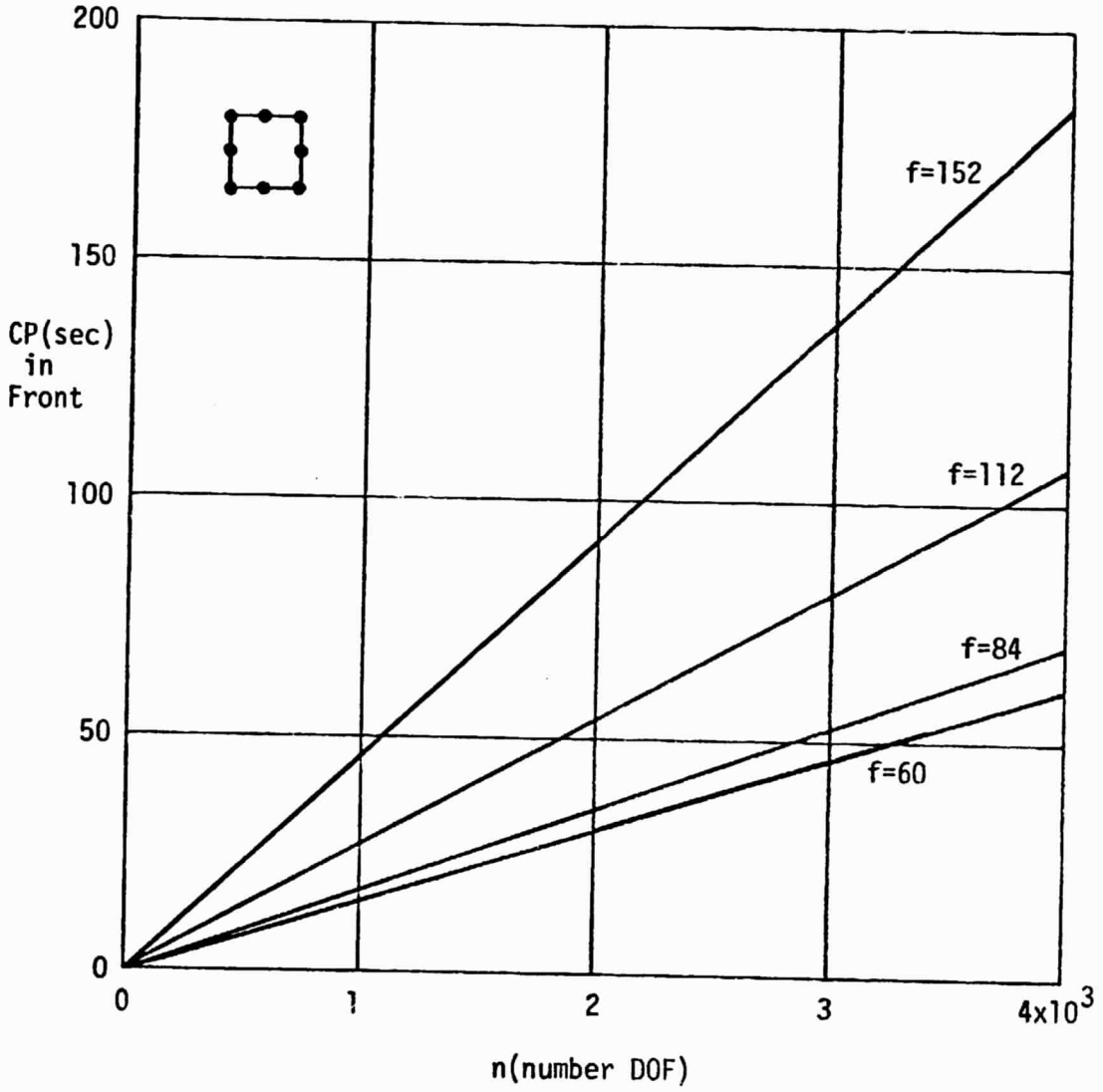


Figure 4.5 CP(sec) in ZIPP Front vs n for Q8

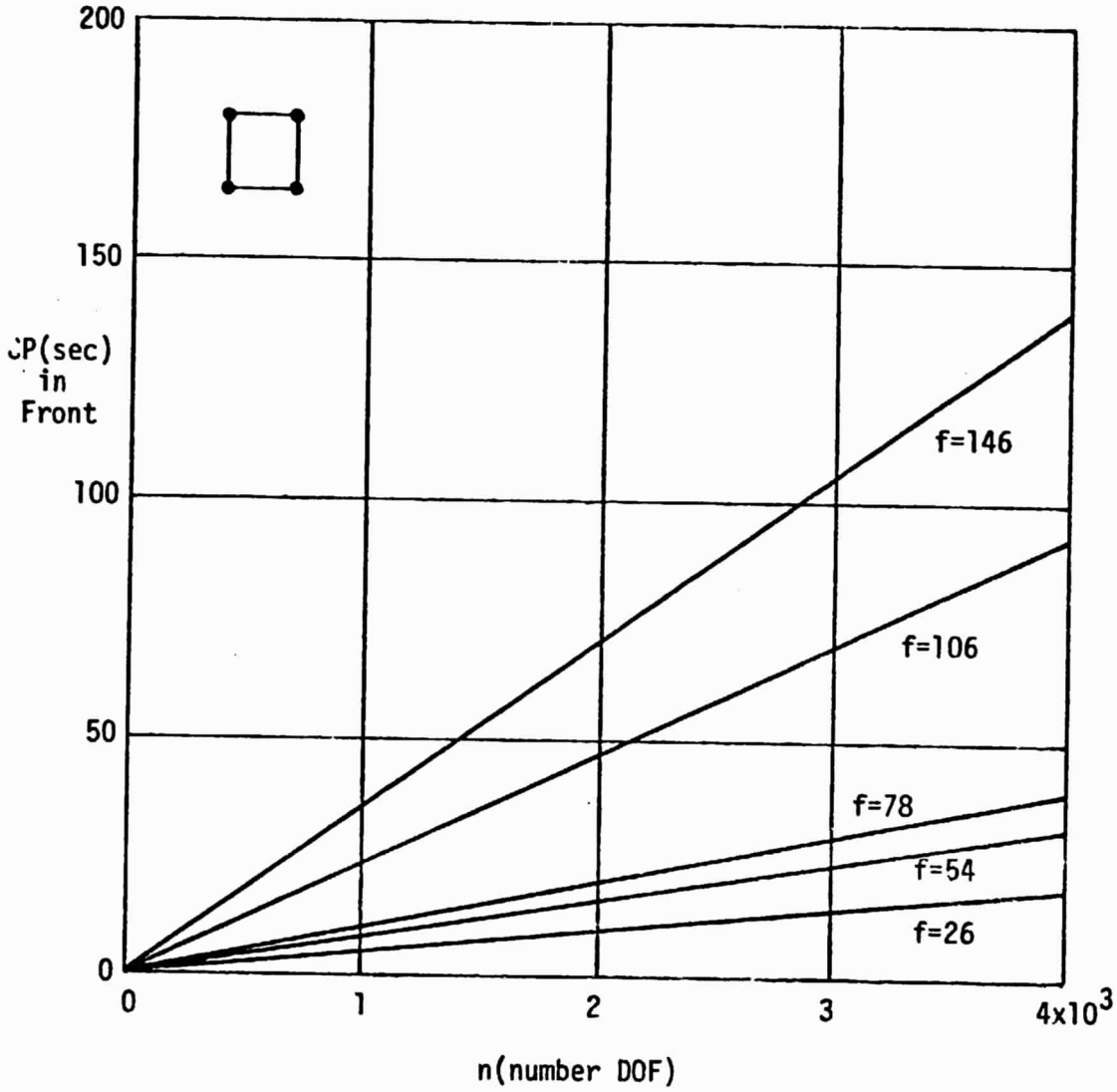


Figure 4.6 CP(sec) in ZIPP Front vs n for Q4

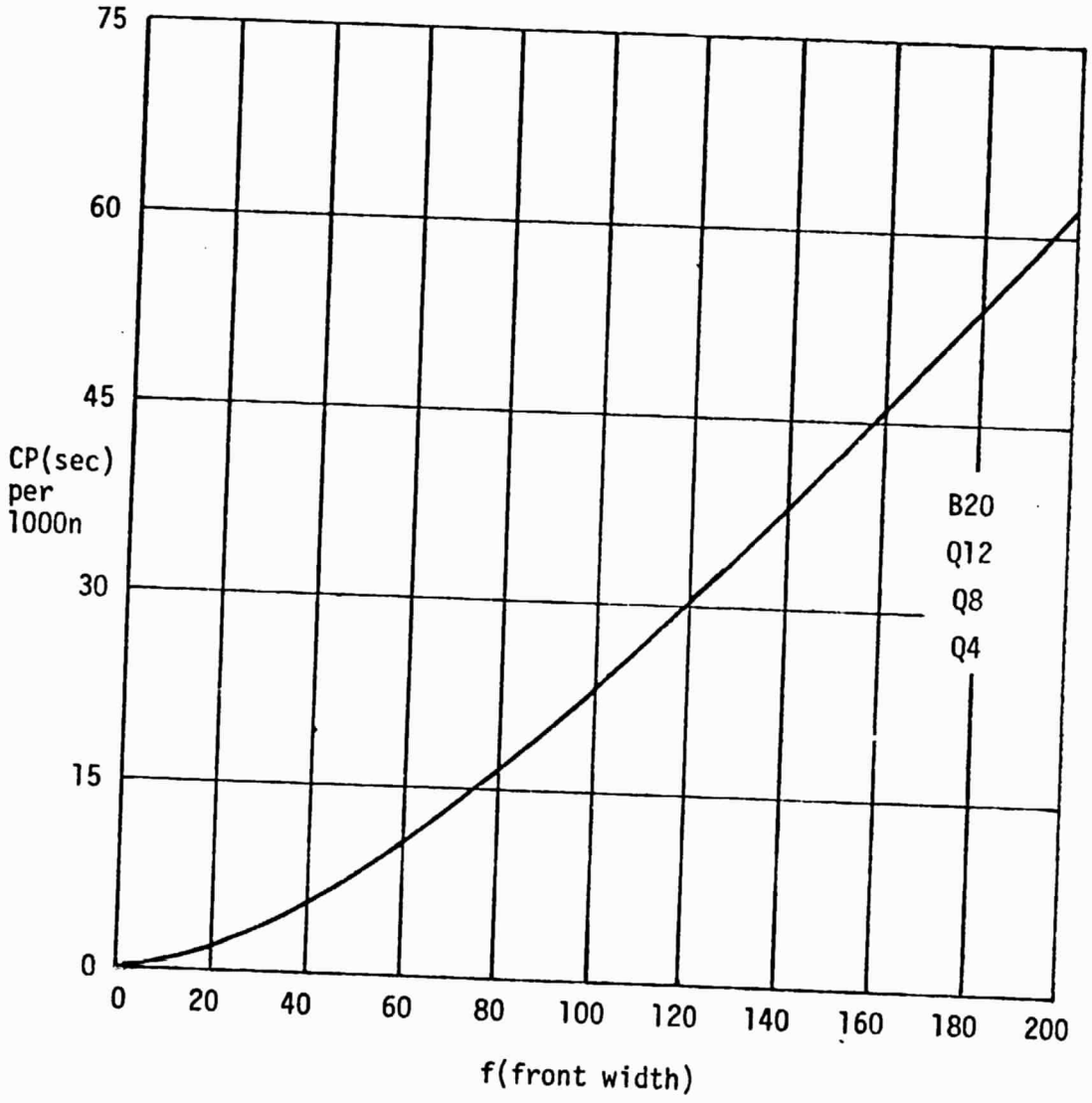


Figure 4.7 ZIPP Front CP Times per 1000 equations

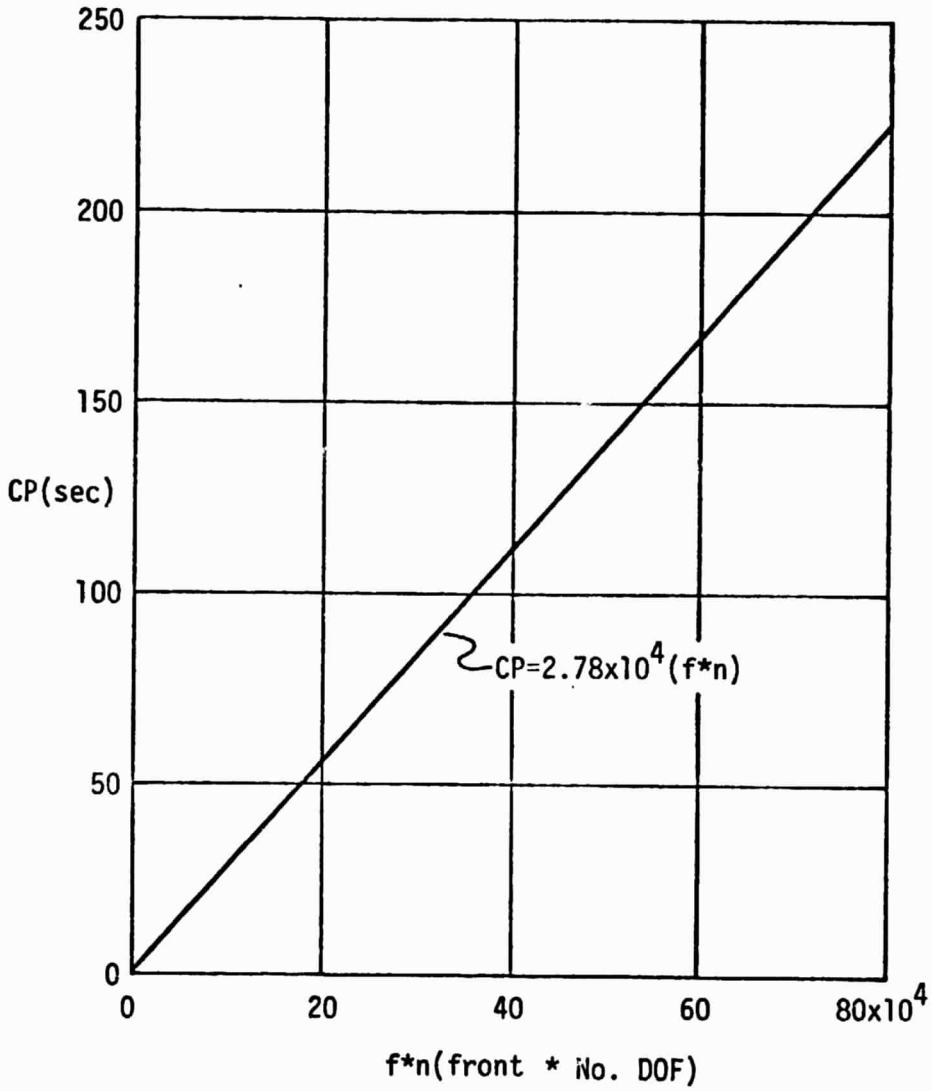


Figure 4.8 ZIPP Front Times vs f*n

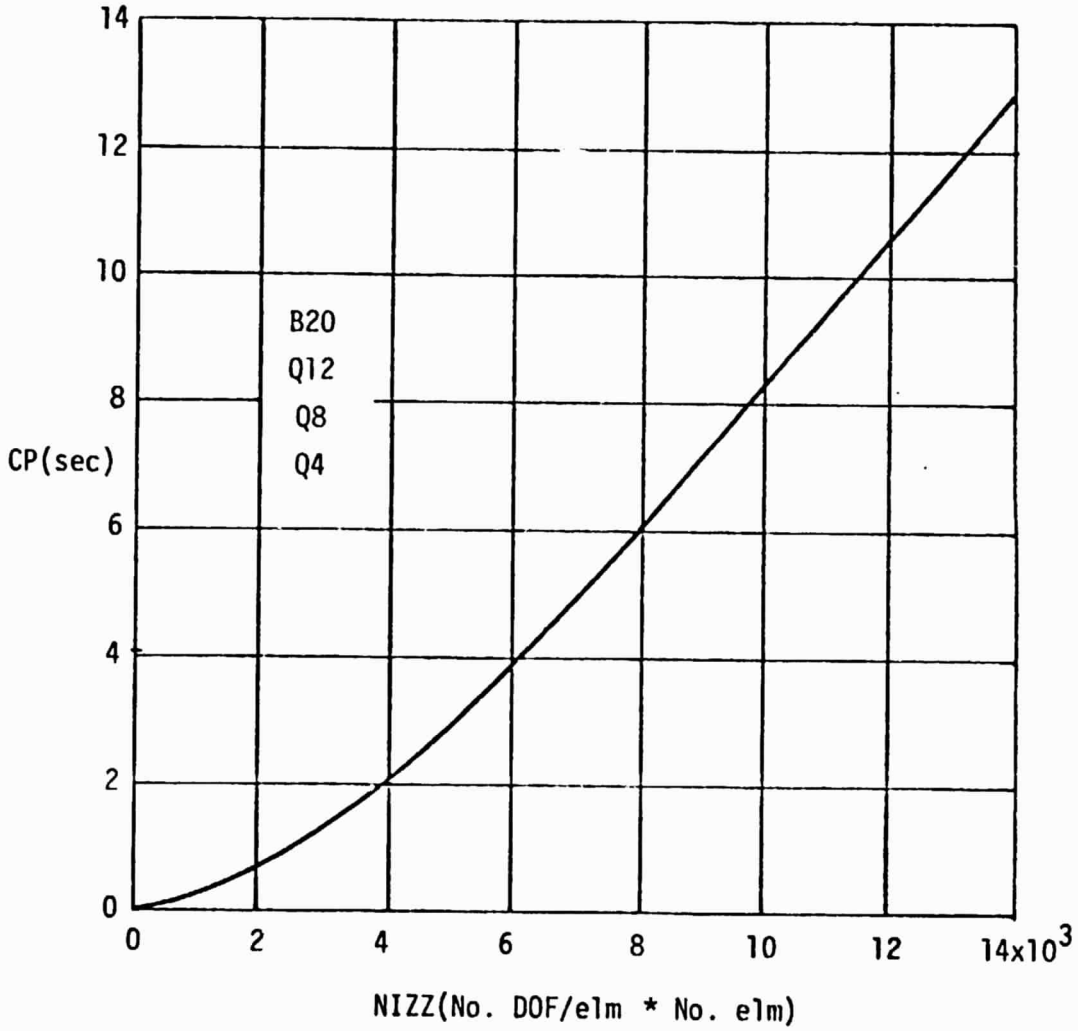


Figure 4.9 ZIPP Backsubstitution T[illegible]es

max/min no. partitions	max words per partition	TM Time (sec) 4x4x3	4x4x2	Δ
1/1	33411	130.8	78.4	52.4
2/1	33153	187.1	78.1	109.0
4/3	14535	289.1	146.3	142.8
16/6	8911	314.3	161.9	152.4

Table 4.9 Total TM Time in PUZZLE for B20 Problems

The first line in Table 4.9 gives the TM times when sufficient core for the entire front is available and gives an incremental time per layer of 4x4 B20's as 52.4 sec. The second line of the table gives the data for a similar run with virtually the same core, but slightly less than needed to fit the front entirely in-core. The column giving the maximum/minimum number of partitions reflects the differences in the number of DOF per element that are eliminated. As more DOF are eliminated, more partitions are needed. In general, the minimum number of partitions reflects the typical situation. The 4x4x2 grid with 2 partitions ran in essentially the same time as 1 partition because with only 2 layers the maximum front is not achieved. However, the 4x4x3 required about 56 sec more TM time. Thus, the increase in Δ TM time was about 100%. The 3 partitions were run with about 15,000 words of core and the increase in Δ time over 1 partition was about 200%. However, the 6 partition run with less than 9000 words of core was almost the same as the 3 partition run. Clearly, PUZZLE is very efficient at handling the extra I/O needed when partitioning the front.

The typical prefront times for all runs (i.e., the rows in Table 4.9) was 10.2 sec for the 4x4x3, 6.5 sec for the 4x4x2 and 3.7 sec for the Δ. The typical backsubstitution time was 16.9 sec for the 4x4x3, 10.7 sec for the 4x4x2 and 6.2 sec for the Δ.

A sequence of calculations were also run on Q8 grids with a variable number of elements across the front(n_f). These results are given in Figure 4.11 where the ratio of PP/CP and TM time per element are plotted versus the total number of elements across the front. The front width is

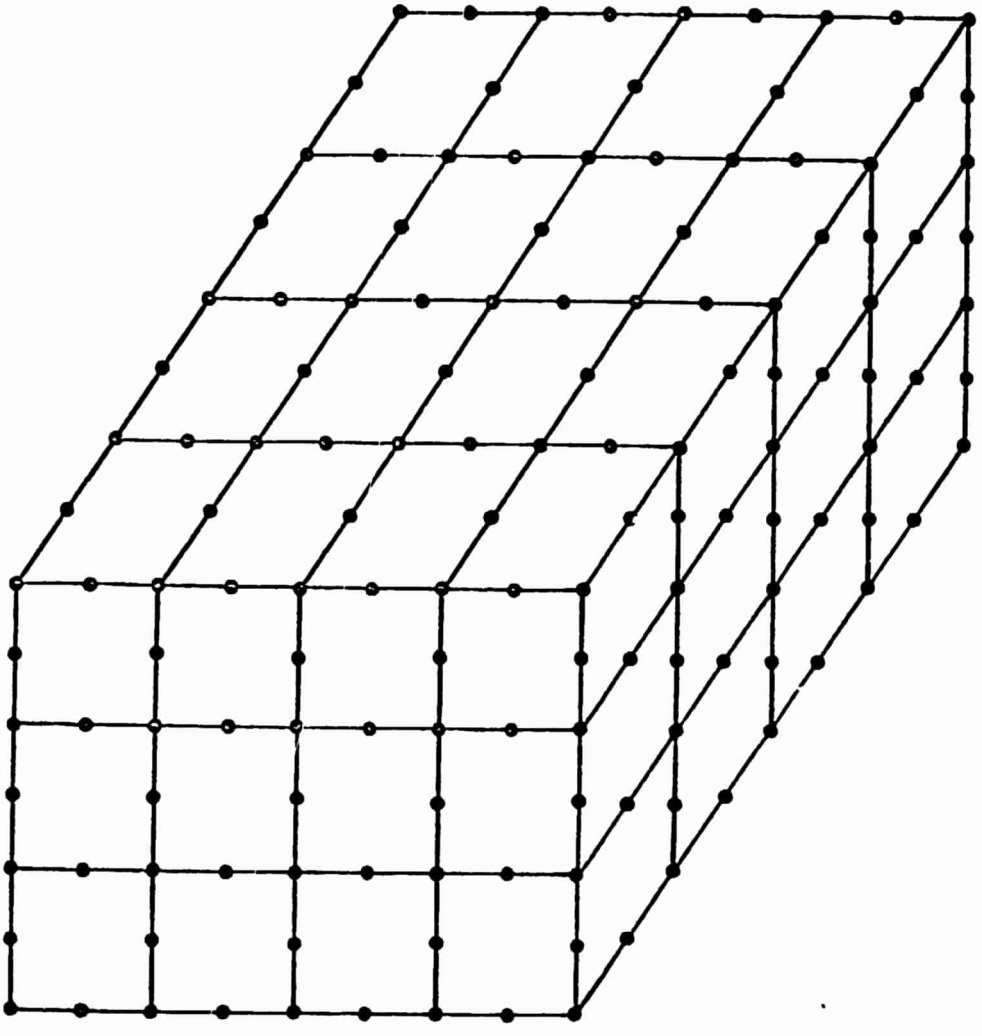


Figure 4.10 4x4x3 Grid of 20-Node Bricks

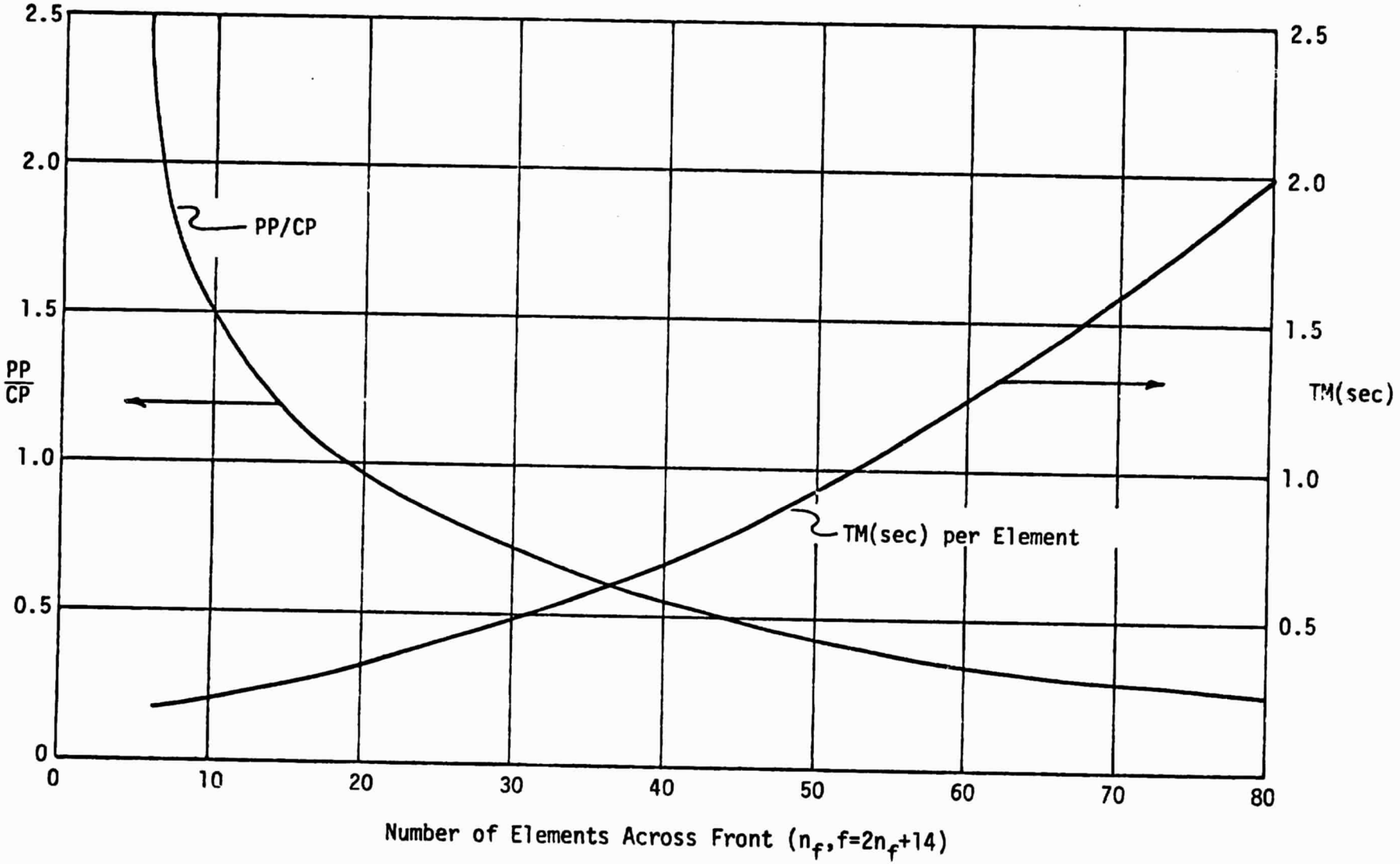


Figure 4.11 PUZZLE Summary Curves for Q8 Element

easily computed from this by the formula

$$f = 2n_f+14$$

Thus, Figure 4.11 gives results for fronts up to f=174. This requires only about 15,000 words of storage and consequently all runs were made without secondary partitioning of the front.

PP stands for the standard CDC "peripheral processing" time. Note TM time equals CP time + a multiple of PP time. In PUZZLE the PP burden is quite low varying in an inverse fashion from 1.5 for n_f=10 to about 0.25 for n_f=80. TM time is seen to increase in a weak quadratic fashion with n_f, varying from 0.20 sec/elm for n_f=10 to 2.0 sec/elm for n_f=80. This is very similar to the behavior reported for ZIPP and again it would be very interesting to see if the weak quadratic behavior continues for larger fronts.

4.5 Qualitative Evaluation of Column Solvers

While the numerical experiments reported herein for the out-of-core column solver COLSOL were done as fairly as possible, these results are undoubtedly less than optimal since no effort was spent enhancing the I/O efficiency of COLSOL. For a more accurate comparison this must be accomplished. However, there are some general qualitative conclusions that can be drawn.

Column solvers generally require the assembly and reduction phases to be separated and this means that a minimum of 4 I/O transfers (2 reads and writes) must be performed. Thus, these column solvers start at a distinct disadvantage to frontal solvers like ZIPP or PUZZLE. Recall that for large fronts or bands the main consideration is I/O efficiency. Also, it is doubtful that central processor variations of more than ±10% will be observed between efficiently coded solvers of either type. Another distinct disadvantage of column solvers is that for regular grids with large fronts or bands, the connectivity between column blocks will always be more

extensive than for frontal blocks because one frontal block will only connect within itself when partitioned, but column connectivities may easily extend across one or more major column blocks.

For these reasons it is doubtful that even the most I/O efficient column solver will ever achieve the low I/O changes reported herein for PUZZLE. This is not to say that column solvers are not useful for many applications where the band is highly variable, but the preliminary evaluation is that for regular 2D and 3D grids (especially with midside nodes), frontal solvers will give lower charges. Notwithstanding, further study of column solvers is clearly indicated.

CHAPTER 5 CONCLUDING REMARKS

The results of this research are probably of questionable value because of the excessive I/O charges on the Texas computer and also because better control over the solvers is undoubtedly necessary. In the present study little attempt was made to recode any of the solvers (other than ZIPP) and the particular patches made to run solvers coded by outside sources are probably not optimal for purposes of careful and accurate comparison. For this reason, the reader is cautioned to make similar studies on the particular compute system being used. However, it is the author's opinion that a great deal of useful qualitative information was learned in the present study (certainly by the author).

Of relevant interest to those who solve large scale 2D and 3D continuum problems on a routine basis are out-of-core frontal and column (skyline) solvers. Clearly in-core and bandsolvers are useful only for testing or research purposes and have no place in large scale general purposes computer programs. There are two subclasses of these solvers of interest, namely, those solvers that partition the front or active column area and those that don't (many column solvers automatically do this). Partitioning is required only when the front becomes too large to fit into available memory. For example, if 4×10^4 words can be dimensioned in central memory then a maximum front of 300 can be handled without partitioning. This would be sufficient for almost all 2D problems and some small 3D problems. However, if for example large memory is used on a CDC 7600, up to 2.5×10^5 can be dimensioned permitting fronts of 700 to be processed. This would be sufficient for all but the largest 3D problems. Thus, users at installations with large systems would typically use ZIPP which does not partition, but on smaller computer systems (e.g., Univ. of Texas, mini or micro-computers) PUZZLE or COLSOL would be necessary for large problems.

The choice between frontal and column solvers has probably not been sufficiently resolved by the present study although the author has some preliminary evaluations. Selection between the two methods should currently be made on the basis of availability, specific adaptation to an individual

system (expecially the I/O), the nature of the code in which the solver is to be used and the particular bias of the user. The frontal method probably offers more flexibility to the programmer doing code development and in particular, ZIPP is quite easy to communicate with. Both frontal and column solvers offer strong potential for substructing and of course substructing invariably lowers computational costs and storage requirements when it can be used. PUZZLE appears to be a very strong competitor in the substructing area. At present the author's choice is the frontal solver, expecially for grids with midside nodes.

The bottom line in evaluation of out-of-core frontal and column solvers appears to be quite simple. If partitioning is not done by the solver, then the I/O charges are fixed. There is no variation between good solvers of either type because the write and read the file of element stiffness coefficients and the file of assembled coefficients exactly once. A solver of this class that does more I/O is unnecessarily inefficient. Therefore, I/O efficiency is not a factor beyond the above point and efficiency can be measured solely on a CP basis. For solvers that partition the front or active column area almost the reverse is true because multiple passes through the element file and the scratch file for the reduced coefficients is required. Since this will typically increase I/O charges by a factor of 2-10, I/O will almost always dominate the charge algorithm. CP efficiency cannot be ignored but variations of 25% would be in the noise level compared to I/O variations. PUZZLE appears to be an extremely strong competitor in the area of I/O efficiency for partitioning solvers. Unlike other schemes, PUZZLE does not require a integer multiple of passes through the scratch file and runs with multiples as low as 1.3 have been observed.

The reader will have undoubtedly detected a bias towards CDC equipment and the author readily admits to such. The 60 bit word effectively eliminates concern over roundoff error for most problems. IBM equipment (32 bits) of course requires full double precision for all computations and Univac equipment (36 bits) likewise requires double precision for almost

all large systems of equations.* Double precision arithmatic is obviously more expensive and for that reason will alter any conclusions about CP efficiency.

While CDC's word size is commendable, its I/O handling is certainly not. There are no standard I/O packages available on CDC equipment and it is this author's experience that each CDC installation has either no such package or four of them; none adequately documented and generally total unknown quantities to the system programmers there. This is an unacceptable situation and at present effectively prohibits the transfer of I/O efficient codes. There are several important factors to consider when improving I/O efficiency. Most I/O routines do some form of buffering that may or may not be useful. Buffering is hardly ever useful and the author recommends direct transfer from central memory to disk without buffering. This is especially important when performing syncronous I/O to random storage (syncronous I/O waits until the transfer is complete before continuing execution). Random storage is generally desirable because the equation solver scratch file must be read in reverse order and because the block size does not need to vary, thus making direct addressing straightforward. Asyncronous I/O (execution continues while I/O is being done) appears attractive due to the obvious CP efficiency. However, in practice to effectively utilize this feature requires a fairly sophisticate coding effort and a tradeoff with the more straightforward syncronous transfer of single large blocks directly (without buffering) to disk may not give a clear advantage to asyncronous schemes. This entire process is future complicated by the CP interruptions on multi-processing computers.

In closing it appears to the author that the question of out-of-core solvers is not a closed one. For static problems the choice seems clearly to go with a frontal or column solver. For implicit dynamic codes this choice is not obvious but the "tilt" would seem to go to iterative or hybrid methods, especially SOR and conjugate gradient techniques. Thus, there appears to be a need for further research. It would seem appropriate to

---

*The smallest recommended word size for our tasks is 48 bits.

continue the study of direct codes but develop frontal and column solver codes from scratch as opposed to using existing subroutines as was done in the present study. The author's choice would be to develop solvers like PUZZLE that partition when necessary but at no loss of efficiency when partitioning is not required. A parallel effort on iterative methods also seems desirable at this time for several reasons. The next generation of computers will permit regular solution of 3D nonlinear dynamic problems. They will also be more prone to I/O binding than present computers simply because the CP will be faster. This will tilt the balance towards number crunching and away from I/O thus making iterative methods more attractive than at present. This latter task is clearly more difficult and the benefits may be slow in being realized.

REFERENCES

1. Dunham, R. S. and Becker, E. B., "TEXGAP-The Texas Grain Analysis Program," TICOM Report 73-1, Univ. of Texas, August 1973.

2. Lehman, F. G., "Simultaneous Equations Solved by Over-Relaxation," Proc 2nd ASCE Conf. Elec. Comp., Pittsburg, September 1960.

3. Young, D. M., "Iterative Solution of Linear and Nonlinear Systems Derived from Elliptic P.D.E.," Lecture Notes in Math., 461, Int. Conf. on Comp. Methods in Nonlinear Mech., September 1974, pp. 265-296.

4. Wilson, E. L., "Finite Element Analysis of Two Dimensional Structures," Ph.D. Dissertation and UC SESM Report 63-2, Univ. of Calif., Berkeley, June 1963.

5. Sequi, W. T., "Computer Programs for the Solution of Systems of Linear Algebraic Equations," NASA CR-2173, MSFC, Ala., January 1973.

6. Schkade, A. F., "Solution Techniques for Large Systems of Stiffness Equations," M.S. Thesis, Univ. of Texas, January 1969.

7. Wilson, E. L., "SOLID SAP A Static Analysis Program for Three Dimensional Solid Structures," UC SESM Report 71-19, September 1971, revised March 1972.

8. Wilson, E. L. and Dovey, H. H., "Solution or Reduction of Equilibrium Equations for Large Complex Structural Systems," to appear (Computers and Structures).

9. Bathe, K. J., "ADINA a Finite Element Program for Automatic Dynamic Incremental Nonlinear Analysis," MIT Acoustics & Vibration Lab Report 82448-1, September 1975.

10. Wilson, E. L., Bathe, K-J and Doherty, W. P., "Direct Solution of Large Systems of Linear Equations," Computers and Structures, V4, 1974, pp. 363-372.

11. Mondkar, D. P. and Powell, G. H., "Large Capacity Equation Solver for Structural Analysis," Computers and Structures, V4, 1974, pp. 699-728.

12. Felippa, C. A., "Solution of Linear Equations with Skyline-Stored Symmetric Matrix," Computers and Structures, V5, 1975, pp. 13-29.

13. Mondkar, D. P. and Powell, G. H., "Toward Optimal In-Core Equation Solving," Computers and Structures, V4, 1974, pp. 531-548.

14. Whetstone, W. D., "Computer Analysis of Large Linear Frames," ASCE Jour. Str. Div., V95, ST11, November 1969, pp. 2401-2417.

15. Goudreau, G. L., Nickell, R. E. and Dunham, R. S., "Plane and Axisymmetric Finite Element Analysis of Locally Orthotropic Elastic Solids and Orthotropic Shells," UC SESM Report 65-15, Univ. of Calif., Berkeley, August 1967.

16. Dunham, R. S. and Nickell, R. E., "Finite Element Analysis of Axisymmetric Solids with Arbitrary Loadings," UC SESM Report 67-6, Univ. of Calif., Berkeley, June 1967.

17. Irons, B. M., "A Frontal Solution Program for Finite Element Analysis," Int. Jour. of Num. Methods Engin., V2, 1970, pp. 5-32.

18. Johnson, C. P., "PUZZLE," Private Communication, Univ. of Texas, 1976.

APPENDIX A

FORTRAN LISTINGS FOR

BANDSOLVERS

APPENDIX A.1

BANSOL BANDED EQUATION SOLVER

PAGE 68 INTENTIONALLY BLANK

```
      PROGRAM BANTEST(INPUT,OUTPUT,TAPE2,TAPE3)
      DIMENSION IY(8),S(256),F(16)
      DATA IDOF/2/,MDIM/12000/,LDIM/864/
      COMMON
     1 MBAND, NNP, ND, NMAX, NUMBLK, B(864), A(12000)
C
      READ 5,NI,NJ,NP
    5 FORMAT(3I5)
      MBAND=(3*NI+5)*IDOF
      NUMEL=NI*NJ
      NNP=(MDIM/MBAND-MBAND)/2
      NUMNP=(2*NI+1)*(NJ+1)+(NI+1)*NJ
      IF(NNP.GT.NUMNP) NNP=NUMNP
      NPD=IDOF*NP
      ND=IDOF*NNP
      IF(ND.LT.MBAND) STOP
      NMAX=ND*MBAND
      NDIM=LDIM+NMAX+MBAND*MBAND
      PRINT 92,ND,NPD,NUMNP,NNP,NUMEL,MBAND
   92 FORMAT(* 92 ND NPD NUMNP NNP NUMEL MBAND*,/,6I10)
      DO 500 N=1,NUMEL
      IF(N.EQ.1) GO TO 3
      DO 90 K=1,NP
      I1=N-1
      I2=I1/NI
      IF(I2*NI.EQ.I1) GO TO 31
      IK=2
      IF(K.EQ.4.OR.K.EQ.8) IK=1
      GO TO 39
   31 IK=NI+4
      IF(K.EQ.4.OR.K.EQ.8) IK=2*(NI+1)+1
   39 IY(K)=IY(K)+IK
   90 CONTINUE
      GO TO 91
    3 READ 20,(IY(I),I=1,NP)
   20 FORMAT(8I5)
   91 K=0
      DO 10 I=1,NPD
      DO 10 J=1,NPD
      K=K+1
      S(K)=1.0E-8
      IF(I.EQ.J) S(K)=1.0
   10 F(I)=1.0
      NPS=NPD*NPD
      NCOUNT=NP+NPS+NPD
      CALL IOP(2HWB,2,IY,NCOUNT)
      IF(N.GT.2) GO TO 500
  500 CONTINUE
      CALL SECOND(T1)
      CALL JOBINFO(8,J1)
      CALL BLOCK(NUMEL,NDIM,NCOUNT,IDOF,NP,NUMNP)
      CALL JOBINFO(8,J2)
      CALL SECOND(T2)
      CP=FLOAT(J2-J1)/1000.
      DT=T2-T1
      PRINT 501,DT,CP
  501 FORMAT(* TOTAL TM TIME IN BANSOL = *,F10.5/
     . * TOTAL CP TIME IN BANSOL = *,F10.5)
      END
      SUBROUTINE BLOCK(NUMEL,NDIM,NCOUNT,IDOF,NP,NUMNP)
      DIMENSION UR(1000)
```

```
      COMMON
     1 MBAND, NNP, ND, NMAX, NUMBLK, B(864), A(12000)
      DIMENSION LM(8),IY(8),S(256),F(16)
C
C*****    FORM STIFFNESS MATRIX IN BLOCKS
C
C
      NUMBLK=0
      DO 50 N=1,NDIM
   50 B(N)=0.0
      ITY=1
      NEL=0
   60 NUMBLK=NUMBLK+1
      NM=NNP*NUMBLK
      NL=NM-NNP+1
      KSHIFT=IDOF*(NL-1)
      IF(NM.GE.NUMNP) ITY=2
C
C*****    SELECT ELEMENT IN BLOCK
C
      CALL IOP(3HREW,2)
      DO 210 N=1,NUMEL
      CALL IOP(2HRB,2,IY,NCOUNT)
      IF(N.LE.NEL) GO TO 210
      DO 80 I=1,NP
      IF (IY(I).LT.NL) GO TO 80
      IF (IY(I).LE.NM) GO TO 90
   80 CONTINUE
      GO TO 210
C
C*****   ADD STIFFNESS AND FORCE VECTOR
C
   90 DO 140 I=1,NP
  140 LM(I)=2*(IY(I)-1)
      NEL=N
      KK=0
      LL=0
      DO 200 I=1,NP
      II=LM(I)-KSHIFT
      IJ=MBAND*(II-1)
      NN=IJ-LM(I)+1
      DO 200 K=1,2
      KK=KK+1
      II=II+1
      IF(N.GT.5) GO TO 181
  181 B(II)=B(II)+F(KK)
      IJ=IJ+MBAND
      NN=NN+MBAND-1
      DO 200 J=1,NP
      JJ=NN+LM(J)
      DO 200 L=1,2
      LL=LL+1
      JJ=JJ+1
      IF (JJ.LE.IJ) GO TO 200
  191 A(JJ)=A(JJ)+S(LL)
  200 CONTINUE
  210 CONTINUE
      NT=IDOF*NUMNP
      CALL BANSOL(UR,ITY)
  390 IF (ITY.NE.2) GO TO 60
      CALL BANSOL(UR,3)
      PRINT 391,(UR(I),I=1,NT)
```

```
  391 FORMAT(* 391 UR(I)*,/(10E12.3))
      RETURN
      END
      SUBROUTINE BANSOL(U,ITY)
C
C*****    1-D BANDED EQUATION SOLVER, REDUCER AND BACKSUBSTITUTER
C
      COMMON
     1 MBAND, NNP, ND, NMAX, NUMBLK, B(864), A(12000)
      DIMENSION U(1), IOTA(30), IOTB(30)
      PRINT 111,ND,NUMBLK,ITY
  111 FORMAT(* 11 1 ND NUMBLK ITY*/,3I10)
      IF (ITY.EQ.3) GO TO 300
C
C*****    REDUCE EQUATIONS AND LOAD VECTOR
C
      N=1-MBAND
      M=0
  110 N=N+MBAND
      M=M+1
      IF (N.GT.NMAX) GO TO 200
      D=A(N)
      IF (D.EQ.0.0) GO TO 110
      I=N
      IJ=N
      MJ=M
      D=1./D
      F=B(M)*D
      B(M)=F
      DO 130 L=2,MBAND
      I=I+1
      IJ=IJ+MBAND
      MJ=MJ+1
      C=A(I)
      IF (C.EQ.0.0) GO TO 130
      B(MJ)=B(MJ)-C*F
      C=C*D
      NK=I
      NJ=IJ
      DO 120 K=L,MBAND
      A(NJ)=A(NJ)-C*A(NK)
      NJ=NJ+1
  120 NK=NK+1
      A(I)=C
  130 CONTINUE
      GO TO 110
  200 IF (ITY.EQ.2) RETURN
C
C*****    WRITE REDUCED BLOCK OF EQUATIONS
C
      IO=IOP(2HWB,3,B,ND)
      IO=IOP(2HWA,3,A,NMAX)
      IO=IOP(2HWR,3)
C
C*****    SHIFT EQUATIONS FOR NEXT BLOCK
C
      N=0
      I=ND
      NJ=0
      IJ=NMAX
  210 N=N+1
      I=I+1
```

```
      B(N)=B(I)
      B(I)=0.0
      DO 220 L=1,MBAND
      NJ=NJ+1
      IJ=IJ+1
      A(NJ)=A(IJ)
  220 A(IJ)=0.0
      IF (N.LT.MBAND) GO TO 210
      N=N+1
      DO 230 I=N,ND
      B(I)=0.0
      DO 230 L=1,MBAND
      NJ=NJ+1
  230 A(NJ)=0.0
      RETURN
C
C*****    BACKSUBSTITUTE FOR UNKNOWNS
C
  300 NK=ND*NUMBLK+1
      NWORDS=ND+NMAX
      NSB=NWORDS/64+1
      NS=NSB*(NUMBLK-1)
  310 N=ND+1
      IJ=NMAX+1
  320 N=N-1
      NK=NK-1
      I=N
      IJ=IJ-MBAND
      NJ=IJ
      F=B(N)
      DO 330 K=2,MBAND
      I=I+1
      NJ=NJ+1
  330 F=F-A(NJ)*B(I)
      B(N)=F
      U(NK)=F
      IF (N.GT.1) GO TO 320
      NUMBLK=NUMBLK-1
      IF (NUMBLK.LE.0) RETURN
C
C*****    SHIFT LAST UNKNOWNS AND READ NEXT BLOCK
C
      I=ND
      N=0
      DO 350 L=1,MBAND
      I=I+1
      N=N+1
  350 B(I)=B(N)
      NS=NS-NSB
      IO=IOP(2HSP,3,NS)
      IO=IOP(2HRB,3,B,ND)
      IO=IOP(2HRB,3,A,NMAX)
      IF(IO.EQ.0) GO TO 310
      PRINT 351
  351 FORMAT(*  ERROR ENCOUNTERED*)
      STOP
      END
```

APPENDIX A.2

USOL BANDED EQUATION SOLVER

```
      PROGRAM SAP2(INPUT=222,OUTPUT=222,TAPE1=1000,TAPE2=1000,TAPE3=1000
     . ,TAPE4=1000,TAPE7=1000,TAPE8=1000,TKPLOT=1000,TAPE5=INPUT,
     . TAPE6=OUTPUT)
C  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  *
C     SAP2 A STATIC ANALYSIS PROGRAM FOR THREE-DIMENSIONAL STRUCTURES
C                          REVISED MARCH 1972
C  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  *
      COMMON A(12000)
      COMMON /KKW/ BB(2000)
      DIMENSION T(7),F(16),LCONEC(16),S(16,16),LM(16)
      DATA LL/1/,IDOF/2/
      DATA F/16*1.0/
C
C     PROGRAM CAPACITY CONTROLLED BY THE FOLLOWING TWO STATEMENTS ...
C
      READ 5 ,NI,NJ,NP
    5 FORMAT(3I5)
      MTOT=12000
      IFL=LOCF(A)
      NUMNP=(2*NI+1)*(NJ+1)+(NI+1)*NJ
      NUMEL=NI*NJ
      NP2=2*NP
      MBAND=IDOF*(3*NI+5)
      NEQ=NUMNP*IDOF
C
C     INPUT JOINT DATA--ID ARRAY ON TAPE 8
C
      N1=1
      N2=N1+6*NUMNP
      N3=N2+NUMNP
      N4=N3+NUMNP
      N5=N4+NUMNP
      N6=N5+NUMNP
      CALL PP(4LRFLP,IFL+MTOT)
C
C     FORM ELEMENT STIFFNESSES--STIFF. ON TAPE 2 -STRESS MATRIX ON TAPE1
C
      REWIND 1
      REWIND 2
      REWIND 4
C
C     INPUT NODAL LOADS AND JOINT MASSES --- WRITE ON TAPE 3
C
      NEQB=(MTOT-4*LL)/(MBAND+LL+1)/2
      NBLOCK=(NEQ-1)/NEQB +1
      IF (NEQB.GT.NEQ) NEQB=NEQ
      N3=N2+NEQB*LL
      N4=N3+6*LL
      WRITE (6,201) NEQ,MBAND,NEQB,NBLOCK
C
C      FORM BB(K) MATRIX, FORM ELEMENT STIFFNESS---ON TAPE 2
C
      ND=2*NP
      LRD=1+ND*(ND+1)
      DO 100 N=1,NUMEL
      IF(N.EQ.1) GO TO 91
      DO 90 K=1,NP
      I1=N-1
      I2=I1/NI
      IF(I2*NI.EQ.I1) GO TO 31
      IK=2
```

PAGE 76 INTENTIONALLY BLANK

```
      IF(K.EQ.4.OR.K.EQ.8) IK=1
      GO TO 39
   31 IK=NI+4
      IF(K.EQ.4.OR.K.EQ.8) IK=2*(NI+1)+1
   39 LCONEC(K)=LCONEC(K)+IK
   90 CONTINUE
      GO TO 92
   91 READ 20,(LCONEC(I),I=1,NP)
   20 FORMAT(8I5)
   92 DO 94 K=1,NP
      K1=2*LCONEC(K)-1
      K2=2*LCONEC(K)
      KR1=2*K-1
      KR2=2*K
      LM(KR1)=K1
      LM(KR2)=K2
      BB(K1)=BB(K1)+F(KR1)
      BB(K2)=BB(K2)+F(KR2)
   94 CONTINUE
      DO 95 I=1,NP2
      DO 95 J=I,NP2
      S(I,J)=1.0E-8
      IF(I.EQ.J) S(I,J)=1.0
   95 CONTINUE
      WRITE(2) LRD,ND,(LM(I),I=1,ND),((S(I,J),J=1,ND),I=1,ND)
  100 CONTINUE
C
C     FORM TOTAL STIFFNESS MATRIX --ON TAPE 4
C
      NE2B=2*NEQB
      N2=N1+NEQB*MBAND
      N3=N2+NEQB*LL
      N4=N3+4*LL
      NN2=N1+NE2B*MBAND
      NN3=NN2+NE2B*LL
      NN4=NN3+4*LL
      CALL SECOND(T(1))
      CALL JOBINFO(8,J1)
      CALL ADDSTF(A(N1),A(NN2),NUMEL,NBLOCK,NE2B,LL,MBAND,NI,NJ,NP)
      CALL JOBINFO(8,J2)
      CALL SECOND(T(2))
C
C     SOLVE FOR DISPLACEMENT UNKNOWNS
C
      NSB=(MBAND+LL)*NEQB
      NSBB=NEQB*LL*(2+(MBAND-1)/NEQB)
      IF(NSBB.LT.NSB) NSBB=NSB
      N4=N3+NSBB
      CALL SECOND(T(3))
      CALL JOBINFO(8,J3)
      CALL USOL(A(N1),A(N3),A(N4),NEQB,MBAND,LL,NBLOCK,NSB,4,3,7,2,2)
      CALL JOBINFO(8,J4)
      CALL SECOND(T(4))
      CSTF=FLOAT(J2-J1)/1000.
      CUSOL=FLOAT(J4-J3)/1000.
      TSTF=T(2)-T(1)
      TUSOL=T(4)-T(3)
      CPTOL=CSTF+CUSOL
      TMTOL=TSTF+TUSOL
      PRINT 30,CSTF,CUSOL,TSTF,TUSOL
   30 FORMAT(* 30 CSTF CUSOL TSTF TUSOL *,4F10.6)
      PRINT 35,CPTOL,TMTOL
```

```
   35 FORMAT(* TOTAL CP TIME IN USOL = *,F10.6/
     . * TOTAL TM TIME IN USOL = *,F10.6)
  201 FORMAT(34H2 TOTAL NUMBER OF EQUATIONS       =,I5,
     1       /34H  BANDWIOTH                      =,I5,
     2       /34H  NUMBER OF EQUATIONS IN A BLOCK =,I5,
     3       /34H  NUMBER OF BLOCKS               =,I5)
      END
      SUBROUTINE MOVEB(ICALL,NEQB,M,NBLOCK,LL,NE2B,U)
      COMMON /KKW/ BB(2000)
      DIMENSION U(NE2B,LL)
      NN=ICALL*NEQB
      DO 1 I=1,NEQB
      DO 1 L=1,LL
      NP=NN+I
      U(I,L)=BB(NP)
    1 CONTINUE
      IF(M.EQ.NBLOCK) RETURN
      DO 2 I=1,NEQB
      DO 2 L=1,LL
      NQ=NP+I
    2 U(NEQB+I,L)=BB(NQ)
      RETURN
      END
      SUBROUTINE ADDSTF (A,B,NUMEL,NBLOCK,NE2B,LL,MBAND,NI,NJ,NP)
C     FORMS GLOBAL EQUILIBRIUM EQUATIONS IN BLOCKS
      DIMENSION A(NE2B,MBAND),SS(1)
      COMMON /EM/ LRD,ND,LM(2592)
      DIMENSION B(NE2B,LL)
      EQUIVALENCE (SS,ND)
      NEQB=NE2B/2
      K=NEQB+1
      X=NBLOCK
      MB=SQRT(X)
      MB=MB/2+1
      NEBB=MB*NE2B
      MM=1
C
      NSHIFT=0
      REWIND 4
C
C     FORM EQUATIONS IN BLOCKS      ( 2 BLOCKS AT A TIME)
C
      ICALL=0
      DO 1000 M=1,NBLOCK ,2
      DO 100 I=1,NE2B
      DO 100 J=1,MBAND
  100 A(I,J)=0.
      CALL MOVEB(ICALL,NEQB,M,NBLOCK,LL,NE2B,B)
      IF (M.EQ.NBLOCK) GO TO 200
      ICALL=ICALL+2
  200 CONTINUE
C
      REWIND 7
      REWIND 2
      NA=7
      NUME=NUM7
      ND=2*NP
      IF(MM.NE.1) GO TO 75
      NA=2
      NUME=NUMEL
      NUM7=0
C
```

```
   75 DO 700 N=1,NUME
      READ(NA) LRD,(SS(I),I=1,LRD)
      DO 600 I=1,ND
      LMN=1-LM(I)
      II=LM(I)-NSHIFT
      IF (II.LE.0.OR.II.GT.NE2B) GO TO 600
      DO 500 J=1,ND
      JJ=LM(J)+LMN
      IF(JJ) 500,500,390
  390 KK=1+ND*J
  400 A(II,JJ)=A(II,JJ)+SS(KK+I)
  500 CONTINUE
  600 CONTINUE
C
C     DETERMINE IF STIFFNESS IS TO BE PLACED ON TAPE 7
C
      IF(MM.GT.1) GO TO 700
      DO 650 I=1,ND
      II=LM(I)-NSHIFT
      IF(II.GT.NE2B.AND.II.LE.NEBB) GO TO 660
  650 CONTINUE
      GO TO 700
  660 WRITE(7) LRD,(SS(I),I=1,LRD)
      NUM7=NUM7+1
C
  700 CONTINUE
      WRITE(4) ((A(I,J),I=1,NEQB),J=1,MBAND),((B(I,L),I=1,NEQB),L=1,LL)
C
      IF(M.EQ.NBLOCK) GO TO 1000
      WRITE(4) ((A(I,J),I=K,NE2B),J=1,MBAND),((B(I,L),I=K,NE2B),L=1,LL)
      IF(MM.EQ.MB) MM=0
      MM=MM+1
 1000 NSHIFT=NSHIFT+NE2B
C
      RETURN
 1002 FORMAT (4F10.0)
 2000 FORMAT(10H2STRUCTURE 12X 25HELEMENT LOAD MULTIPLIERS /
     . 10H LOAD CASE 9X 1HA 9X 1HB 9X 1HC 9X 1HD/)
 2002 FORMAT (I6,7X,4F10.3)
      END
      SUBROUTINE USOL (A,B,MAXB,NEQB,MB,LL,NBLOCK,NSB,NORG,NBKS,NT1,
     .                                                    NT2,NRST)
      DIMENSION A(NSB),B(NSB),MAXB(NEQB)
C-----------------------------------------------------------------------
      NC=MB+LL
      NBR=(MB-1)/NEQB+1
      INC=NEQB-1
      NMB=NEQB*MB
      N2=NT2
      N1=NT1
      REWIND NORG
      REWIND NBKS
C
C---- REDUCE EQUATIONS  BLOCK-BY-BLOCK ---------------------------------
C
      DO 900 N=1,NBLOCK
      IF (N.GT.1.AND.NBR.EQ.1) GO TO 110
      IF (NBR.EQ.1) GO TO 105
      REWIND N1
      REWIND N2
  105 NI=N1
      IF(N.EQ.1) NI=NORG
```

```
      READ (NI) A
  110 DO 300 I=1,NEQB
      D=A(I)
      IF(D) 115,300,120
  115 M=NEQB*(N-1)+I
      WRITE (6,116) M,D
  116 FORMAT (33H0SET OF EQUATIONS MAY BE SINGULAR   /
     .  26H DIAGONAL TERM OF EQUATION   I8, 8H  EQUALS   1PE12,4)
C
  120 II=I
      DO 125 J=2,NC
      II=II+NEQB
  125 A(II)=A(II)/D
C
      DO 130 J=I,NMB,NEQB
      IF (A(J).NE.0.) MAXB(I)=J
  130 CONTINUE
C
      JL=I+1
      IF (JL.GT.NEQB) GO TO 300
      II=I
      DO 200 J=JL,NEQB
      II=II+NEQB
      IF(II.GT.NMB) GO TO 200
      C=A(II)
      IF (C.EQ.0.0) GO TO 200
      C=C*A(I)
C
      KK=J-II
      MAX=MAXB(I)
      DO 150 JJ=II,MAX,NEQB
  150 A(JJ+KK)=A(JJ+KK)-C*A(JJ)
C
      KK=J +NMB
      JJ=I+NMB
      DO 175 L=1,LL
      A(KK)=A(KK)-C*A(JJ)
      KK=KK+NEQB
  175 JJ=JJ+NEQB
  200 CONTINUE
  300 CONTINUE
      WRITE (NBKS) A,MAXB
C
C---- SUBSTITUTE INTO REMAINING EQUATIONS ----------------------------
C
      DO 800 NN=1,NBR
      IF(N+NN.GT.NBLOCK) GO TO 800
      NI=N1
      IF(N.EQ.1) NI=NORG
      IF(NN.EQ.NBR) NI=NORG
      READ (NI) B
      IL=1+NN*NEQB*NEQB
      DO 700 I=1,NEQB
      II=IL
      DO 690 K=1,NEQB
      IF (II.GT.NMB) GO TO 690
      C=A(II)
      IF (C.EQ.0.0) GO TO 690
      C=C*A(K)
      MAX=MAXB(K)
C
      KK=I-II
```

```
      DO 640 JJ=II,MAX,NEQB
  640 B(JJ+KK)=B(JJ+KK)-C*A(JJ)
C
      KK=I+NMB
      JJ=K+NMB
      DO 650 L=1,LL
      B(KK)=B(KK)-C*A(JJ)
      KK=KK+NEQB
  650 JJ=JJ+NEQB
C
  690 II=II-INC
  700 IL=IL+NEQB
C
      IF(NBR.NE.1) GO TO 750
      DO 740 I=1,NSB
  740 A(I)=B(I)
      GO TO 800
  750 WRITE (N2) B
  800 CONTINUE
C
      M=N1
      N1=N2
  900 N2=M
C
C---- BACKSUBSTITUTION - RESULTS ON TAPE NRST  -----------------------
C
      LS=LL*NEQB
      NEB=NEQB*(NBR+1)
      NUM=NBR*NEQB
      MAX=NEB*LL
      DO 905 I=1,MAX
  905 B(I)=0.
      REWIND NRST
C-------------------------------------------------------------------
      DO 1000 N=1,NBLOCK
      BACKSPACE NBKS
      READ (NBKS) A,MAXB
      BACKSPACE NBKS
      DO 910 L=1,LL
      K=L*NEB
      DO 910 J=1,NUM
      I=K-NEQB
      B(K)=B(I)
  910 K=K-1
C
      I=NMB
      DO 920 L=1,LL
      K=(L-1)*NEB
      DO 920 J=1,NEQB
      I=I+1
      K=K+1
  920 B(K)=A(I)
C
      DO 955 I=1,NEQB
      J=NEQB+1-I
      MAX=MAXB(J)
      IF (A(J).EQ.0.) GO TO 955
      DO 950 L=1,LL
      KK=J+(L-1)*NEB
      JJ=KK+1
      IL=J+NEQB
      C=B(KK)
```

```
      DO 940 II=IL,MAX,NEQB
      C=C+A(II)*B(JJ)
  940 JJ=JJ+1
  950 B(KK)=C
  955 CONTINUE
C
      I=0
      DO 960 L=1,LL
      K=(L-1)*NEB
      DO 960 J=1,NEQB
      K=K+1
      I=I+1
  960 A(I)=B(K)
C
      WRITE (NRST) (A(I),I=1,LS)
      PRINT 970,(A(I),I=1,LS)
  970 FORMAT(* 970 A(I)*/,(10E12.3))
 1000 CONTINUE
C-----------------------------------------------------------------------
      RETURN
      END
```

APPENDIX B

FORTRAN LISTINGS OF

FRONTAL SOLVERS

APPENDIX B.1

ZIPP FRONTAL EQUATION SOLVER

```
      PROGRAM HSMAIN ( INPUT,OUTPUT,TAPE13,TAPE14,TAPE15,TAPE16,TAPE19,
     . TAPE5=INPUT,TAPE6=OUTPUT)
      COMMON /FORIER/ NUMEL
      COMMON /TAPES/ NIN,NOUT,NTAPE1,NTAPE2,NTAPE3,NTAPE4,NTAPE5,
     . NTAPE6,NTAPE7,NTAPE8,NTAPE9
      COMMON/ZIP/LVABL(28),KUREL,LPREQ,LZ,LDEST(28),DUM(5),MVABL(160)
      COMMON EL(25100)
      COMMON /IOINFO/ JPRINT
      DIMENSION LCONEC(28)
C
C     *** NFUNC(I,J) DEFINED ***
      NFUNC(I,J)=I+(J*(J-1))/2
C
    1 FORMAT(1H1///,*     ----------  NEW PROBLEM  ----------* )
   10 FORMAT ( 2I5,I10 )
   15 FORMAT( ///5X,*NUMEL=*,I5,5X,*NEWRHS=*,I5,5X,*NELPAZ=*,I10 )
   20 FORMAT ( I5,/,(10I5) )
   51 FORMAT(* 51 CONECTIVITY*,(8I5))
   30 FORMAT ( I5,/,(5F10.0) )
      JPRINT = 6
      NTAPE3 =13
      NTAPE4 =14
      NTAPE5 =15
      NTAPE6 =16
      NTAPE9 =19
      NIN= 5
      NOUT = 6
 1000 CONTINUE
      IO=HSIO(NTAPE3,10,LVABL,0,0)
      IO=HSIO(NTAPE4,10,ELPA,0,0)
      PRINT 1
      READ 10,NI,NP
      READ  10, NUMEL, NEWRHS ,NELPAZ
      PRINT 15, NUMEL, NEWRHS ,NELPAZ
      IF ( NUMEL .LE. 0 )  STOP
      READ 20,KUREL,(LCONEC(I),I=1,NP)
      NP2=2*NP
      LZ=KUREL*(KUREL+3)/2
      DO  100  N = 1, NUMEL
      IF(N.EQ.1) GO TO 91
      DO 90 K=1,NP
      I1=N-1
      I2=I1/NI
      IF(I2*NI.EQ.I1) GO TO 31
      IK=2
      IF(K.EQ.4.OR.K.EQ.8) IK=1
      GO TO 39
   31 IK=NI+4
      IF(K.EQ.4.OR.K.EQ.8) IK=2*(NI+1)+1
   39 LCONEC(K)=LCONEC(K)+IK
   90 CONTINUE
      IF(N.GT.12) GO TO 91
      PRINT 51,(LCONEC(K),K=1,NP)
   91 DO 95 J=1,NP
      J1=2*J-1
      J2=2*J
      LVABL(J1)=2*LCONEC(J)-1
      LVABL(J2)=2*LCONEC(J)
   95 CONTINUE
  100 IO=HSIO(NTAPE4,1,LVABL,29,0)
      DO 101 I=1,LZ
```

PAGE 88 INTENTIONALLY BLANK

```
      EL(I)=1.0E-8
  101 CONTINUE
      DO 102 I=1,NP2
      NF=NFUNC(I,I)
      EL(NF)=1.0
  102 CONTINUE
      LZK=LZ-KUREL
      DO 103 I=LZK,LZ
      EL(I)=1.0
  103 CONTINUE
      DO 200 N=1,NUMEL
  200 IO=HSIO(NTAPE3,1,EL,LZ,1)
      IO=HSIO(NTAPE3,10,LVABL,0,0)
      IO=HSIO(NTAPE4,10,EL,0,0)
      CALL SECOND(TK)
      CALL JOBINFO(8,J1)
      CALL ZIPP(NELPAZ)
      CALL JOBINFO(8,J2)
      CALL SECOND (TW)
      CP=FLOAT(J2-J1)/1000.
      DTKW=TW-TK
      PRINT 201,DTKW,CP
  201 FORMAT(* 201 TM IN ZIPP = *,F10.6/5X,*CP IN ZIPP = *F10.6)
      GO TO 1000
      END
      SUBROUTINE ZIPP(NELPAZ)
C
C.....    FRONTAL SOLUTION METHOD BY BRUCE IRONS
C
      COMMON /FORIER/ NUMEL
      COMMON /TAPES/ NIN,NOUT,NTAPE1,NTAPE2,NTAPE3,NTAPE4,NTAPE5,
     . NTAPE6,NTAPE7,NTAPE8,NTAPE9
      COMMON/ZIP/LVABL(28),KUREL,LPREQ,LZ,LDEST(28),DUM(5),MVABL(160)
      COMMON /IOINFO/ JPRINT
      COMMON ELPA(1)
      DIMENSION NIX(1)
      EQUIVALENCE (NIX,ELPA)
C
      NFUNC(I,J)=I+(J*(J-1))/2
      CALL SECOND(T0)
      CALL JOBINFO(8,J0)
      NBIG=1000
      NELZ=462
      NIZZ =0
      MAXPA=0
      NVABZ=0
      LCUREQ=0
      NIXEND=NELPAZ
      LVEND=28
      MVEND=160
      DO 232 I=1,MVEND
  232 MVABL(I)=0
      IO=HSIO(NTAPE6,10,ELPA,0,0)
      IO=HSIO(NTAPE4,10,ELPA,0,0)
      IO=HSIO(NTAPE5,10,ELPA,0,0)
C
C     PUT ALL ELEMENT NICKNAMES IN LONG VECTOR, NIX
C
      JWHERE=5
      DO  10  NELEM = 1, NUMEL
      IO=HSIO(NTAPE4,2,LVABL,29,1)
      IF(NIZZ+KUREL+NELEM.GT.NIXEND) GO TO 130
```

```
      DO 8 I=1,KUREL
      NIZZ=NIZZ+1
    8 NIX(NIZZ)=-LVABL(I)
      NIX(NIXEND+1-NELEM)=NIZZ
   10 CONTINUE
      N1=1
      DO 26 NELEM=1,NUMEL
      LPREQ=LCUREQ
      LCUREQ=NVABZ
      NZ=NIX(NIXEND+1-NELEM)
      KUREL=NZ-N1+1
      LZ=NFUNC(2*KUREL,KUREL)
C
C     FIND NEW NICKNAMES AND USE EXISTING DESTINATIONS.
C
      DO 22 NEW=N1,NZ
      NIC=NIX(NEW)
      LDES=NIC
      IF(NIC.GT.0) GO TO 20
      DO 14 LDES=1,MVEND
      IF(MVABL(LDES).EQ.0) GO TO 16
   14 CONTINUE
      JWHERE=7
      GO TO 130
   16 MVABL(LDES)=NIC
      IF(LDES.GT.MAXPA) MAXPA=LDES
      KOUNT=1
C
C     RECORD FIRST, LAST AND INTERMEDIATE APPEARANCES.
C
C
C*****    THE NEXT FIVE(5) STATEMENTS SHOULD BE REPLACED WHEN
C*****    CONVERTING THE COMPASS ROUTINE FIRLAS TO FORTRAN
C
      NND=NIZZ-NEW+1
      NICD=NIC
      LDESD=LDES
      CALL FIRLAS(NND,NICD,NIX(NEW),LDESD,LAST,KOUNT)
      LAST=LAST+NEW-1
C
      KOUNT=KOUNT*NBIG
      NIX(LAST)=LDES + NBIG
      LDES=LDES+KOUNT
      NIX(NEW)=LDES
   20 LDEST(NEW-N1+1)=LDES
   22 CONTINUE
      N1=NZ+1
C
C     RECONSTRUCT ELEMENT NICKNAMES AND COUPLE WITH DESTINATION VECTORS,
C     AND INITIAL ELEMENT STIFFNESS AND LOAD DATA.
C
      DO 24 KL=1,KUREL
      CALL CODEST(LDES,LDEST(KL),NSTRES,NBIG)
      LVABL(KL)=-MVABL(LDES)
      IF(NSTRES.NE.0.AND.NSTRES.NE.1) GO TO 24
      MVABL(LDES)=0
      NVABZ=NVABZ+1
   24 CONTINUE
C
C *** REWRITE ALL ELEMENT INFORMATION ON TAPE
C
      IO=HSIO(NTAPE5,1,LVABL,64,0)
```

```
   26 CONTINUE
      IO=HSIO(NTAPE5,9)
      CALL JOBINFO(8,J1)
      CALL SECOND(T1)
      CP=FLOAT(J1-J0)/1000.
      DT=T1-T0
      WRITE(NOUT,870) DT,CP,NIZZ,NVABZ,MAXPA
  870 FORMAT(*0TIME IN PREFRONT = *F8.3//
     .       *  CP  IN PREFRONT = *F8.3//
     .       * NIZZ  = *I5//
     .       * NVABZ = *I5//
     .       * MAXPA = *I5)
      IO=HSIO(NTAPE5,10,ELPA,0,0)
      IO=HSIO(NTAPE3,10,ELPA,0,0)
C
C     PRE-PROGRAM ENDED AND ELEMENT TAPE WRITTEN.
C
C     ESTABLISH STORAGE REQUIREMENTS AND AREA BOUNDARIES.
C
      NPAR=NFUNC(0,MAXPA+1)+NELZ
      NPAZ=NPAR+MAXPA
      NBAXO=NPAZ+1
      NBUFFA=NELPAZ-NBAXO
      JWHERE=9
      IF(NBUFFA.LT.MAXPA+4) GO TO 130
      NRUNO=NPAZ-MAXPA
      IBA=NBAXO
      DO 38 I=1,NELPAZ
      ELPA(I)=0.0
   38 CONTINUE
      KURPA=0
C
C    SEEK AND ASSEMBLE NEW ELEMENT.
C
      IPOS=0
      NWORDS=0
      DO 92 NELEM=1,NUMEL
      IO=HSIO(NTAPE5,2,LVABL,64,1)
      IO=HSIO(NTAPE3,2,ELPA,LZ,1)
      L=0
      DO 40 KL=1,KUREL
      CALL CODEST(LDES,LDEST(KL),NSTRES,NBIG)
      MVABL(LDES)=LVABL(KL)
      LVABL(KL)=LDES
      IF(LDES.GT.KURPA) KURPA=LDES
   40 CONTINUE
      DO 64 LHSRHS=1,2
      LHS=2-LHSRHS
       IRHS=1-LHS
       MNM=LHS*KUREL+IRHS
      DO 64 KL=1,MNM
      GO TO (42,44),LHSRHS
   42 KG=LVABL(KL)
      MGO=NFUNC(0,KG)+NELZ
      GO TO 46
   44 MGO=(KL-1)*MAXPA+NPAR
   46 MMN=LHS*KL+IRHS*KUREL
      DO 64 IL=1,MMN
      IG=LVABL(IL)
      L=L+1
      CE=ELPA(L)
      MG=MGO+IG
```

```
      IF(LHSRHS.EQ.1.AND.KG.LT.IG) MG=NFUNC(KG,IG)+NELZ
      ELPA(MG)=ELPA(MG) + CE
   64 CONTINUE
C
C     ELIMINATE VARIABLE IN POSITION LDES, AND WRITE EQUATION FOR TAPE
C
      DO 90 KL=1,KUREL
      CALL CODEST(LDES,LDEST(KL),NSTRES,NBIG)
      IF(NSTRES.NE.0.AND.NSTRES.NE.1) GO TO 90
      NDEQN=IBA+KURPA+4
      IF(NDEQN.LE.NELPAZ) GO TO 70
      ELPA(IBA+1)=NWORDS
      NWORDS=IBA-NBAX0+2
      NS=NWORDS/64
      IF(64*NS.NE.NWORDS) NS=NS+1
      IPOS=IPOS+NS
      IF(JPRINT.GE.6) WRITE(NOUT,3002) NELEM,NWORDS,IPOS,NBAX0,IBA,TEMP
 3002 FORMAT(* INTERMEDIATE TAPE6 WRITE *,5I10,E15.4)
      IO=HSIO(NTAPE6,1,ELPA(NBAX0),64*NS,1)
      IBA=NBAX0
      NDEQN=IBA+KURPA+4
   70 IBDIAG=IBA+LDES
      NDIAG=NFUNC(0,LDES+1) +NELZ
      PIVOT=ELPA(NDIAG)
      ELPA(NDIAG)=0.0
      JWHERE=13
      NIC=MVABL(LDES)
      IF(PIVOT.EQ.0.) GO TO 130
      MGZ=NELZ
      JGZ=KURPA
      IB0=IBA
      DO 84 LHSRHS=1,2
      IF(LHSRHS.EQ.2) JGZ=1
      DO 84 JG=1,JGZ
      IBA=IBA+1
      GO TO (72,76),LHSRHS
   72 MG0=MGZ
      MGZ=MG0+JG
      IF(LDES.GT.JG) GO TO 74
      MG=MG0+LDES
      GO TO 78
   74 MG=NFUNC(JG,LDES)+NELZ
      GO TO 78
   76 MG0=(JG-1)*MAXPA+NPAR
      MG=MG0+LDES
      MGZ=MG0+KURPA
   78 NDELT=IB0-MG0
      CONST=ELPA(MG)
      ELPA(IBA)=CONST
      IF(CONST.EQ.0) GO TO 84
      CONST=CONST/PIVOT
      ELPA(MG)=0.0
      IF (LHSRHS.NE.1) GO TO 80
C
C     SIMULTANEOUSLY, CREATE A SIMPLE ROUND OFF CRITERION,
C
   80 NNL=MG0+1
C
C*****     THE CALL INNER SHOULD BE REPLACED WHEN
C*****      CONVERTING FROM COMPASS TO FORTRAN
C
      CALL INNER(MG0,MGZ,CONST,ELPA(NNL),NDELT)
```

```
   84 CONTINUE
      ELPA(IBDIAG)=PIVOT
      IBA=NDEQN
      ELPA(IBA)=KURPA
      ELPA(IBA-1)=LDES
      ELPA(IBA-2)=MVABL(LDES)
   86 MVABL(LDES)=0
      IF(MVABL(KURPA),NE,0) GO TO 90
      KURPA=KURPA-1
      IF(KURPA,NE,0) GO TO 88
   90 CONTINUE
   92 CONTINUE
      IO=HSIO(NTAPE6,9)
      IO=HSIO(NTAPE6,10)
      CALL JOBINFO(8,J2)
      CALL SECOND(T2)
      CP=FLOAT(J2-J1)/1000.
      DT=T2-T1
      WRITE(NOUT,871) DT,CP,IPOS,NPAZ
  871 FORMAT(*0TIME IN FRONT = *F8.3//
     .       *  CP  IN FRONT = *F8.3//
     .       * NUMBER OF SECTORS = *I5//
     .       * NPAZ  = *I5)
C
C     BACK-SUBSTITUTE INTO EQUATIONS, TAKEN IN REVERSE ORDER.
C
      NBZ=IBA
      NEQ=NVABZ
      LPREQ=LCUREQ
      NELEM=NUMEL
      IO=HSIO(NTAPE9,10)
  100 IF(IBA,NE,NBAXO) GO TO 102
      NS=NWORDS/64
      IF(64*NS,NE,NWORDS) NS=NS+1
      IPOS=IPOS-NS
      NBZ=NWORDS+NBAXO-2
      IO=HSIO(NTAPE6,6,ELPA,IPOS,0)
      IO=HSIO(NTAPE6,2,ELPA(NBAXO),64*NS,1)
      NWORDS=ELPA(NBZ+1)
      IBA=NBZ
  102 KURPA=ELPA(IBA)
      LDES=ELPA(IBA-1)
      NIC=ELPA(IBA-2)
      IBAR=IBA-4
      IBA=IBAR-KURPA
      IBDIAG=IBA+LDES
      PIVOT=ELPA(IBDIAG)
      ELPA(IBDIAG)=0.0
      MGO=NRUNO
      MGZ=MGO+KURPA
      CONST=ELPA(IBAR+1)
      NDELT=IBA-MGO
      NNL=MGO+1
C
C*****    THE CALL BSUB SHOULD BE REPLACED WHEN
C*****    CONVERTING FROM COMPASS TO FORTRAN
C
      CALL BSUB(NNL,MGZ,CONST,ELPA(NNL),NDELT)
C
C     PLACE ANSWERS AND PREPARE FOR NEW ITERATIVE LOOP.
C
      ELPA(MGO+LDES)=CONST/PIVOT
```

```
      ELPA(IBDIAG)=PIVOT
      NEQ=NEQ-1
  108 IF(NEQ.NE.LPREQ) GO TO 100
      IO=HSIO(NTAPE5,6,LVARL,NELEM-1,0)
      IO=HSIO(NTAPE5,2,LVARL,64,1)
      DO 112 KL=1,KUREL
      CALL CODEST(LDES,LDEST(KL),NSTRES,NBIG)
      NRUN=NRUNO+LDES
      ELPA(KL)=ELPA(NRUN)
  112 CONTINUE
      ELPA(64)=KUREL
      IO=HSIO(NTAPE9,1,ELPA(1),64,1)
      NELEM=NELEM-1
      IF(NELEM.NE.0) GO TO 108
      IO=HSIO(NTAPE9,9)
      CALL JOBINFO(8,J3)
      CALL SECOND(T3)
      CP=FLOAT(J3-J2)/1000.
      CP3=FLOAT(J3-J0)/1000.
      DT=T3-T2
      T3=T3-T0
      WRITE(NOUT,872) DT,CP,T3,CP3
  872 FORMAT(*0TIME IN BSUB = *F8.3//
     .       *  CP  IN BSUB = *F8.3//
     .       * TOTAL TIME IN ZIPP = *F8.3//
     .       * TOTAL  CP IN ZIPP = *F8.3)
      RETURN
  130 CONTINUE
      WRITE(NOUT,834) JWHERE,NIC,PIVOT,LDES,LZ,NELZ,NELEM,
     1   NBUFFA,NIZZ,NELPAZ,MAXPA,NPAZ
      CALL ERROR(JWHERE,0,0,10HZIPPJWHERE)
  850 FORMAT(19H0MAX FRONT WIDTH = ,I3,22H MAX ACTIVE STORAGE = ,I5,
     1 19H MAX NIX STORAGE = ,I5)
  834 FORMAT(9H0JWHERE =,I3,5X,5HNIC =,I8,5X,
     1   7HPIVOT =,E12.4,3X,5HLDES=,I5,3X,4HLZ =,I5,11X,6HNELZ =,I5/
     2   8H NELEM =,I4,5X,8HNBUFFA =,I6,4X,
     3   6HNIZZ =,I5,9X,8HNELPAZ =,I5,4X,
     4   9H MAXPA = ,I4,10H   NPAZ = ,I7)
      END
      SUBROUTINE CODEST(LDES,LDEST,NSTRES,NBIG)
      NS=LDEST/NBIG
      LDES=LDEST-NS*NBIG
      NSTRES=NS-1
      RETURN
      END
      FUNCTION HSIO(NTAPE,NOP,A,NWORDS,NWAIT)
C
C.....    CDC 6600/6400 HIGH SPEED BINARY I/O FORTRAN INTERFACE ROUTINE
C.....    NWAIT OPTION NOT OPERATIONAL ON CDC, BUT IT IS USEFUL ON
C.....    UNIVAC 1108 SERIES WITH THEIR NTRAN ROUTINE.
C.....    NWAIT=0 FOR NO WAIT  ;  NWAIT=1 FOR WAIT TO COMPLETE I/O
C
      COMMON/TAPES/NIN,NOUT
      INTEGER HSIO
      IF(NOP-2) 1,2,3
C
C.....    WRITE NWORDS STARTING AT ADDRESS A
C
    1 HSIO=IOP(2HWB,NTAPE,A,NWORDS)
      GO TO 90
C
C.....    READ NWORDS STARTING FROM ADDRESS A
```

```
C
      2 HSIO=IOP(2HRB,NTAPE,A,NWORDS)
        IF(HSIO,EQ,0) GO TO 90
C
C.....    ERROR DIAGONSTIC FOR INCOMPLETE I/O TRANSFER
C
        WRITE(NOUT,3000) NTAPE,NOP,NWORDS,NWAIT,HSIO
 3000 FORMAT(*0HSIO READ ERROR / NTAPE NOP NWORDS NWAIT HSIO*//
     .  ,6I5)
        CALL ERROR(2,0,0,4HHSIO)
      3 IF(NOP-9) 6,9,10
C
C.....    SET POSITION TO SECTOR NUMBER NWORDS - USED FOR BACKSPACE
C
      6 HSIO=IOP(2HSP,NTAPE,NWORDS)
        GO TO 90
C
C.....    WRITE END-OF-FILE MARK
C
      9 HSIO=IOP(2HWF,NTAPE)
        GO TO 90
C
C.....    REWIND
C
     10 HSIO=IOP(3HREW,NTAPE)
     90 RETURN
        END
```

```
          IDENT INNER                    (MGO,MGZ,CONST,ELPA,NDELT,)
          ENTRY    INNER
INNER     BSSZ     1
*
*         COMPASS VERSION OF THE FOLLOWING LOOP
*
*        DO 82 I=NNL,MGZ
*        ELPA(I)=ELPA(I) - CONST*ELPA(I+NDELT)
*    82  CONTINUE
*
*     INITIALIZE
*
          SB7      1              STORE 1 INTO B7
          SA3      B3             CONST TO X3
          SA1      B1             MGO TO X1
          SB1      X1+B7           MGO TO B1
          SA2       B2              MGZ TO X2
          SB3       X2              MGZ TO B3
          SA4      B4             ELPA(MGO+1) TO X4
          SA5      B5             NDELT TO X5
          SB5      X5             NDELT TO B5
          SA5      A4+B5           ELPA(I+NDELT) TO X5
          RX7      X3*X5          FIRST MULT, CONST*ELPA(I+NELT)
          BX0      X4             COPY X4 INTO X0
           GE       B1,B3,EXIT     SKIP LOOP IF MGO+1=MGZ
*
*    INNER LOOP
*
LOOP      FX6      X0-X7            ELPA-CONST*ELPA(I+NDELT)
           NX6      X6             NORMALIZE RESULT IN X6
           SA6      A4             STORE RESULT IN ELPA(I)
           SA4      A4+B7          NEXT ELPA(I) TO X4
           SA5      A5+B7          NEXT ELPA(I+NDELT) TO X5
          RX7      X3*X5          CONST* ELPA(I+NDELT)
          BX0      X4             ELPA(I) TO X4
```

```
          SB1       B1+B7           ADD 1 TO INCR COUNTER
          LT        B1,B3,LOOP      TEST FOR END OF LOOP
*
*     LAST SUBSTRACTION
*
EXIT       FX6       X0-X7
          NX6       X6                NORMALIZE X6
           SA6       A4               STORE LAST RESULT IN ELPA(I)
          EQ        B0,B0,INNER       RETURN TO PROG.
          END
          IDENT     BSUB              (NNL,MGZ,CONST,ELPA(NNL),NDELT)
          ENTRY     BSUB
BSUB      BSSZ      1
*
*         COMPASS VERSION OF THE FOLLOWING LOOP
*
*         DO 104 I=NNL,MGZ
*         CONST=CONST - ELPA(I)*ELPA(I+NDELT)
*     104 CONTINUE
*
*     INITIALIZE
*
          SB7       1                  STORE 1 INTO B7
          SA3       B3                CONST TO X3
          SA1       B1                NNL TO X1
          SB1       X1                NNL TO B1
          SA2       B2                MGZ TO X2
          SB2       X2                MGZ TO B2
          SA4       B4                ELPA(I) TO X4
          SA5       B5                NDELT TO X5
          SB5       X5                NDELT TO B5
          SA5       A4+B5             ELPA(I+NDELT) TO X5
          RX7       X4*X5             FIRST MULTIPLY
          BX0       X3                 COY CONST TO X0
          GE        B1,B2,EXIT        SKIP LOOP IF NNL =MGZ
*
*     INNER LOOP
*
LOOP      FX6       X0-X7             CONST=ELPA(I)*ELPA(I+NDELT)
          NX6       X6                NORMALIZE RESULT IN X6
          SA4       A4+B7             NEW ELPA TO X4
          SA5       A5+B7             NEW ELPA(I+NDELT) TO X5
          RX7       X4*X5              ELPA(I)*ELPA(I+NDEL)
          BX0       X6                CONST TO X0  X0
          SB1       B1+B7              ADDS ONE TO COUNTER
          LT        B1,B2,LOOP        TEST FOR END OF LOOP
*     LAST SUBSTRACTION
EXIT      FX6       X0-X7             LAST SUBXTRACTION
          NX6       X6                NORMALIZE RESULT IN X6
          SA6       A3                STORE RESULT IN CONST
          EQ        B0,B0,BSUB        RETURN
          END
          IDENT     FIRLAS               (NND,NICD,NIX(NEW),LDES,LAST,KOUNT
          ENTRY     FIRLAS
*
*          COMPASS VERSION OF THE FOL. LOOP
*
*          KOUNT=1                                  ORIGINAL PAGE IS
*          DO 18 LAS=NEW,NIZZ                       OF POOR QUALITY
*          IF(NIX(LAS).NE.NIC) GO TO 18
*          NIX(LAS)=LDES
*          LAST=LAS
```

```
*           KOUNT=KOUNT + 1
*      18   CONTINUE
*
*           INITIALIZATION
*
FIRLAS      BSSZ    1
*
            SA1     B1                  NND TO X1
            SB1     X1                  NND TO B1
            SB7     1                   1 TO B7
            SA2     B2                  NIC TO X2
            SA3     B3                  NIX(1) TO X3
            SA4     B4                  LDES TO X4
            BX6     X4                  COPY LDES IN X6
            SB2     1                   SET B2=1
            SB4     B2                 KOUNT=1
*
LOOP        IX7     X3-X2               NIX(LAS)-NIC TO X7
            NZ      X7,INCR             IF NIX(LAS),NE,NIC JUMP
            SA6     A3
            SB2     B7                  LAST=LAS
            SB4     B4+1
INCR        SB7     B7+1               INCREMENT LOOP COUNT BY 1
            SA3     A3+1               NEXT NIX (LAS)
            LE      B7,B1,LOOP          IF(LAS,LE,NND)  LOOP
*
            SX6     B2                  B5 TO X6
            SA6     B5                  LAST TO B5
            SX7     B4                  KOUNT TO B6
            SA7     B6
*
            EQ      B0,B0,FIRLAS        RETURN
            END
      SUBROUTINE ERROR(NERR,I,J,WORD)
      COMMON /TAPES/ NIN,NOUT
      IF(I,NE,0) GO TO 5
      WRITE(NOUT,10) NERR,WORD
      STOP 1
    5 WRITE(NOUT,10) NERR,WORD,I,J
   10 FORMAT(* ERROR NUMBER = *,I6,2X,A10,2I6)
      STOP 1
      END
```

APPENDIX B.2

PUZZLE FRONTAL EQUATION SOLVER

C-2

```
      PROGRAM PUZZLE (INPUT,OUTPUT,TAPE1,TAPE2,TAPE3,TAPE4,TAPE5,TAPE6)
      COMMON           NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / 0000 /  MAXF,MAXB,MAXW,IPRNT
      COMMON / TRAC /  NWS,NWR,NBS,NUP

      CALL SECOND ( T1 )

      CALL JOBINFO(8,J1)

      CALL SETDATA

      CALL JOBINFO(8,J2)
      CALL SECOND ( T2 )

      CALL FRONTIQ

      CALL JOBINFO(8,J3)
      CALL SECOND ( T3 )

      CALL FRONTST

      CALL JOBINFO(8,J4)
      CALL SECOND ( T4 )

      CALL BACKSUB

      CALL JOBINFO(8,J5)

      CALL SECOND ( T5 )

      TQ = T2 - T1
      CP=FLOAT(J2-J1)/1000.
      PRINT 100,TQ,CP
      TQ = T3 - T2
      CP=FLOAT(J3-J2)/1000.
      PRINT 200,TQ,CP
      TQ = T4 - T3
      CP=FLOAT(J4-J3)/1000.
      PRINT 300,TQ,CP
      TQ = T5 - T4


      CP=FLOAT(J5-J4)/1000.
      PRINT 400,TQ,CP
      TQ = T5 - T1
      CP=FLOAT(J5-J1)/1000.
      PRINT 500,TQ,CP

      PRINT 600,NWS,NWR
      PRINT 700,NBS,NUP

  100 FORMAT ( *1* 5X *TIME FOR SETDATA* F13.3 /
     .             6X * CP  FOR SETDATA* F13.3)
  200 FORMAT (     6X *TIME FOR FRONTIQ* F13.3 /
     .             6X * CP FOR FRONTIQ* F13.3 )
  300 FORMAT (     6X *TIME FOR FRONTST* F13.3 /
     .             6X * CP  FOR FRONTST* F13.3 )
  400 FORMAT (     6X *TIME FOR BACKPAS* F13.3 /
     .             6X *  CP FOR BACKPAS* F13.3 )
  500 FORMAT (     6X *TIME FOR PUZZLE * F13.3 /
```

```
     *              6X * CP  FOR PUZZLE * F13.3 )
600 FORMAT (       6X *NWS * I10 / 6X *NWR * I10 )
700 FORMAT (       6X *NBS * I10 / 6X *NUP * I10 )

    END
    SUBROUTINE SETDATA
    COMMON             NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
    COMMON / 0000 /    MAXF,MAXB,MAXW,IPRNT
    COMMON / SUBT /    NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
    COMMON / PRNT /    IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
    COMMON / MESH /    IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ
    COMMON / SCR1 /    IR,LP,I1,I2,NR,MD,NW,S(2041)
    COMMON / TRAC /    NWS,NWR,NBS,NUP
    LP  = 0
    NWS = 0
    NWR = 0
    NBS = 0
    NUP = 0


    DO 1 I = 1,6
  1 CALL WIND ( I )

    READ  800,MAXF,MAXB,MAXW
    PRINT 900,MAXF
    PRINT 901,MAXB
    PRINT 902,MAXW
    READ  100,IGO,IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
    PRINT 100,IGO,IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
    READ  100,NOPT,NOEL,NDFRE,NODES,NDIVY,NDIVX,NDIVZ,IPRNT

    IF ( IGO.EQ.1 ) CALL SETU1
    IF ( IGO.EQ.2 ) CALL SETU2
    IF ( IGO.EQ.3 ) CALL SETU3
    IF ( IGO.EQ.4 ) CALL SETU4
    IF ( IGO.EQ.5 ) CALL SETU5

    PRINT 200,NOPT ,NOEL ,NDFRE
    PRINT 201,NODES,NDIVY,NDIVX
    PRINT 202,NDIVZ,IPRNT,LP

    CALL WIND ( 1 )
    CALL WIND ( 2 )

100 FORMAT (20I4)
200 FORMAT ( *1* 5X *NOPT * I5 / 6X *NOEL * I5 / 6X *NDFRE* I5 )
201 FORMAT (     6X *NODES* I5 / 6X *NDIVY* I5 / 6X *NDIVX* I5 )
202 FORMAT (     6X *NDIVZ* I5 / 6X *IPRNT* I5 / 6X *IPOS * I5 )
800 FORMAT (  4I8    )
900 FORMAT ( *1* 5X *MAXF SET AT* I8 )
901 FORMAT (     6X *MAXB SET AT* I8 )
902 FORMAT (     6X *MAXW SET AT* I8 )

    RETURN
    END
    SUBROUTINE SETU1
    COMMON             NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
    COMMON / 0000 /    MAXF,MAXB,MAXW,IPRNT
    COMMON / SUBT /    NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
    COMMON / MESH /    IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ

                  READ  100,(NDOF(I),I=1,NOPT)
```

```
      IF(IPRNT.GE.2) PRINT 100,(NDOF(I),I=1,NOPT)

      DO 1 I=1,NOEL
                     READ  100,NODES,ME,(IQ(L),L=1,NODES)
      IF(IPRNT.GE.2) PRINT 300,NODES,ME,(IQ(L),L=1,NODES)

      CALL UNIT1 (1,1,NODES,IQ)

    1 CONTINUE

      DO 4 LZ = 1,NOEL
                     READ  100,ME
      IF(IPRNT.GE.2) PRINT 100,ME
                     READ  500,(PE (M),M=1,ME)
      IF(IPRNT.GE.2) PRINT 500,(PE (M),M=1,ME)
      DO 2 I = 1,ME
                     READ  500,(ST(I,J),J=1,ME)
    2 IF(IPRNT.GE.2) PRINT 500,(ST(I,J),J=1,ME)

      CALL UNIT2 ( ME,PE,ST )
    4 CONTINUE

  100 FORMAT (20I4)
  300 FORMAT (6X,*JQ*,9I5)
  500 FORMAT ( 20F6.0 )
      RETURN
      END
      SUBROUTINE SETU2
      COMMON            NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
      COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / MESH /   IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ

      DO 1 I = 1,NOPT
    1       NDOF(I) = NDFRE

      DO 2 K=1,NOEL
      READ  100,NODES,ME,(IQ(L),L=1,NODES)
      CALL READY
      CALL UNIT2 ( ME,PE,ST )
    2 CONTINUE

  100 FORMAT (10I4)

      RETURN
      END
      SUBROUTINE SETU3
      COMMON            NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
      COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / MESH /   IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ
      DIMENSION JQ(4)

           NOEL = NDIVY * NDIVX
           NOPT = (NDIVY+1)*(NDIVX+1)

      DO 1 I  = 1,NOPT
    1       NDOF(I) = NDFRE

           ME = NDFRE * NODES
           NP1 = NDIVY + 1
      DO 3 IX  = 1,NDIVX
           JQ(1) = (IX-1) * NP1 + 1
           JQ(4) = JQ(1)        + 1
```

```
      JQ(2) = JQ(1)  + NP1
      JQ(3) = JQ(4)  + NP1
  DO 3 IY  = 1,NDIVY
  DO 2 I   = 1,4
2     IQ(I) = JQ(I) + IY - 1
      CALL READY
       CALL UNIT2 ( ME,PE,ST )
3      CONTINUE
  RETURN
  END
  SUBROUTINE SETU4
  COMMON            NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
  COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
  COMMON / MESH /   IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ
  DIMENSION JQ (8)

      NOEL = NDIVY * NDIVX
      NOPT = 3*NDIVY*NDIVX + 2*(NDIVY+NDIVX) + 1

  DO 1 I  = 1,NOPT
1     NDOF(I) = NDFRE

      ME     = NDFRE * NODES
      NP1     = NDIVY + 1
      INC     = 3 * NDIVY + 2

      JQ(1)   = 1
      JQ(8)   = 2
      JQ(4)   = 3
      JQ(5)   = 2 * NP1
      JQ(7)   = 2 * NP1 + 1
      JQ(2)   = 3 * NP1
      JQ(6)   = JQ(2)   + 1
      JQ(3)   = JQ(2)   + 2
  DO 5 IX  = 1,NDIVX
      L1  = 0
      L2  = 0
  DO 3 IY  = 1,NDIVY
  DO 2 I   = 1,4
2     IQ(I)  = JQ(I) + L2

      IQ(5)  = JQ(5) + L1
      IQ(6)  = JQ(6) + L2
      IQ(7)  = JQ(7) + L1
      IQ(8)  = JQ(8) + L2

      L1  = L1        + 1
      L2  = L2        + 2
      CALL READY
      CALL UNIT2 ( ME,PE,ST )
3     CONTINUE
  DO 4 I   = 1,NODES
4     JQ(I)   = JQ(I) + INC

5     CONTINUE

  RETURN
  END
  SUBROUTINE SETU5
  COMMON            NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
  COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
  COMMON / MESH /   IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ
```

```
      DIMENSION JQ(20)
            NOEL = NDIVX * NDIVY * NDIVZ
            N1   = (2*NDIVY+1)*(NDIVX+1) + (NDIVY+1)*NDIVX
            N2   = (  NDIVY+1)*(NDIVX+1)
            N1N2 = N1 + N2
            NOPT = N1 + N1N2 * NDIVZ
      DO  1 I = 1,NOPT
    1       NDOF(I) = NDFRE
            ME      = NDFRE * NODES
      DO  2 I = 1,3
    2       JQ( I) = I
            L = 2 + 3*NDIVY
      DO  3 I = 1,3
    3       JQ(I+3) = I + L
      DO  4 I = 1,6
    4       JQ(I+6) = JQ(I) + N1N2
            JQ(13) = 2*NDIVY + 2
            JQ(14) = JQ(13) + 1
            JQ(15) = JQ(13) + N1N2
            JQ(16) = JQ(15) + 1
            JQ(17) =    N1  + 1
            JQ(18) =    N1  + 2
            JQ(19) = JQ(18) + NDIVY
            JQ(20) = JQ(19) + 1

      DO 11 IZ = 1,NDIVZ
            L1 = 0
            L2 = 0
            L3 = 0
      DO  9 IX = 1,NDIVX
      DO  8 IY = 1,NDIVY
      DO  5 I  = 1,12
    5       IQ(I) = JQ(I) + L1
      DO  6 I = 13,16
    6       IQ(I) = JQ(I) + L2
      DO  7 I = 17,20
    7       IQ(I) = JQ(I) + L3
            L1 = L1 + 2
            L2 = L2 + 1
            L3 = L3 + 1
            CALL READY
            CALL UNIT2 ( ME,PE,ST )
    8 CONTINUE
            L1 = ( 3*NDIVY+2 ) * IX
            L2 = L1
            L3 = L3 + 1
    9 CONTINUE

      DO 10 I = 1,20
   10       JQ(I) = JQ(I) + N1N2

   11 CONTINUE
      RETURN
      END
      SUBROUTINE READY
      COMMON             NDOF(2500),IQ(20),ME,PE(60),ST(60,60)
      COMMON / MESH /  IGO,NDFRE,NODES,NDIVY,NDIVX,NDIVZ
      COMMON / 0000 /  MAXF,MAXB,MAXW,IPRNT

      IF(IPRNT.GE.2) PRINT 100,NODES,ME
      IF(IPRNT.GE.2) PRINT 200,(IQ(L),L=1,NODES)
```

```
      CALL UNIT1 (1,1,NODES,IQ)

      DO 1 I = 1,ME
    1   PE(I)= 0.0
          I3     = NDOF(1)
          L      = 0
      DO 2 I = 1,ME,I3
          L      = L + 1
          K2     = I + I3 - 1
      DO 2 K = I,K2
    2   PE(K) = IQ(L)

      DO 3 I = 1,ME
      DO 3 J = 1,ME
    3 ST(I,J)= 1.0E-20

      DO 4 I = 1,ME
    4 ST(I,I)= PE(I)

  100 FORMAT ( 6X  2I5 )
  200 FORMAT ( 6X 12I5 )
      RETURN
      END
      SUBROUTINE UNIT2 ( ME,PE,ST )
      DIMENSION PE(60),ST(60,60)
      COMMON / SCR1 /  IR,LP,I1,I2,NR,MD,NW,S(2041)
      COMMON / 0000 /  MAXF,MAXB,MAXW,IPRNT
      COMMON / TRAC /  NWS,NWR,NBS,NUP
          MD = ME
          NW = (MD*(MD+1))/2 + MD + 7
          IF ( NW.GT.MAXB )                                  GOTO 1
          I1 =  1
          I2 = MD

                                                             GOTO 6
    1     I2 =  0
          IT = MD + 1

    2     I1 =  I2 + 1
          I2 =  I1
          NW =   7
      DO 3 I  =  I1 , MD
          NW =  NW + IT
          IF  ( NW.GT.MAXB )                                 GOTO 4
          I2 =  I2 + 1
    3     IT =  IT - 1

                                                             GOTO 5
    4     NW =  NW - IT
    5     I2 =  I2 - 1
    6     NR =  I2 - I1 + 1
          L  = 0
      DO 7 I  = I1 , I2
          L  = L  +  1
    7   S(L) = PE(I)

      DO 8 I  = I1 , I2
      DO 8 J  =  I , MD
          L  = L  +  1
    8   S(L) = ST (I,J)

                        IR    = NW / 64
                        IS    = IR * 64
      IF( IS.NE.NW )    IR    = IR +  1
```

```
                    NW    = IR *64 + 7
                    LP    = IR + LP

          CALL IOP (2HWB,2,  IR,   7)
          CALL IOP (2HWB,2,   S,  NW)
          NWS = NWS + NW + 7

          IF ( I2.LT.MD )                                    GOTO 2

      RETURN
      END
      SUBROUTINE FRONTIQ
      COMMON              NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
      COMMON              NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
      COMMON              NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
      COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT
      DATA L1,L5 /1,5/

          READ   5,MSUB
          PRINT 10,MSUB
          IM2 = MSUB + 1

          CALL SECOND   ( T1 )

          ISAV = NOEL
          NSTR = 0
      DO 1 IM = 1,IM2
          NT   =   0
    1     CALL SUBIQS   ( NT,IM )

      DO 2 IM = 1,6
    2     CALL WIND ( IM )

    5 FORMAT ( I4 )
   10 FORMAT ( *1* 5X *NUMBER OF SUBSTRUCTURES IS* I5 )
      RETURN
      END
      SUBROUTINE SUBIQS ( NT,IM )
      COMMON              NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
      COMMON              NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
      COMMON              NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
      COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT
      DATA L1,L5 /1,5/

          IF (IM.LE.MSUB)                                    GOTO 1
          CALL LOCATE ( NOEL,IM,NT,L1,L1 )
                                                             GOTO 5

    1                 READ  10,   NSUB,(ISUB(I),I=1,NSUB)
          IF(IPRNT.GE.2) PRINT 20,IM,NSUB,(ISUB(I),I=1,NSUB)

          CALL WRIT3 (1)

          NUEL = 0
      DO 4  IT   = 1,NSUB
                 READ  10,NODE2,(IQ2(L),L=1,NODE2)
          IF(IPRNT.GE.2) PRINT 30,NODE2,(IQ2(L),L=1,NODE2)

      DO 2  I  = 1,NODE2
```

```
          L  =      IQ2(I)
 2        NDOF(L) =-NDOF(L)

          LSUB = ISUB (IT)
          CALL WIND ( 6 )
          CALL LOCATE ( LSUB,IM,NT,L1,6 )
          NUEL =  NUEL + LSUB

   DO 3   I = 1,NOPT
 3        NDOF(I)  = IABS(NDOF(I))

 4        CALL UNIT1 ( L5,1,N2,MOC2 )

          CALL MERGEIQ ( L1,L5,NODE2,IQ2)

10 FORMAT ( 20I4 )
20 FORMAT (*0* 5X *IM   * I4 / 6X *NSUB* I4 / 6X *ISUB* 20I4 )
30 FORMAT (    6X*MQ    *22I4 )

 3 RETURN
   END
   SUBROUTINE MERGEIQ ( L1,L5,NODES,IQ )
   DIMENSION IQ(1)
   COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)

        NOEL = NOEL - NUEL
        IF ( NOEL.EQ.0 )                    GOTO 2

        DO 1 I = 1,NOEL

        CALL UNIT1 ( L1,2,NODES,IQ )
        CALL UNIT1 ( L5,1,NODES,IQ )

1       CONTINUE

2       NOEL = NOEL + NSUB
        NSTR = NSTR + NSUB

        CALL WIND ( L1 )
        CALL WIND ( L5 )

        LT  =  L1
        L1  =  L5
        L5  =  LT

        RETURN
        END
  SUBROUTINE LOCATE ( LSUB,IM,NT,L1,LT )
  COMMON            NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
  COMMON            NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
  COMMON            NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
  COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)

        NQL = 0

  DO 1 I    = 1,NOPT
1       IB(I) = 0

  DO 4 LZ   = 1,LSUB

        CALL UNIT1 ( L1,2,NODE2,IQ2 )
        IF (IM.GT.MSUB)
```

ORIGINAL PAGE IS
OF POOR QUALITY

GOTO 2

```
      CALL UNIT1 ( LT,1,NODE2,IQ2 )

2 DO 3 I   = 1,NODE2
      K    = IQ2(I)
3     IB(K) = IB(K) + 1
4     CONTINUE

  DO 5 I   = 1,NOPT
5     IC(I) = IB(I)

      CALL WIND ( LT )

      LB = LSUB - 1
      CALL FRONT  ( NT,LT,0, 0 )
      IF ( LB.EQ.0 )                                    GOTO 7

  DO 6 LZ = 1,LB
                                KZ = LB - LZ
      CALL UPDATE
      CALL FRONT  ( NT,LT,0,KZ )
6     CALL EXPAND ( LZ         )

7     CALL UPDATE
      CALL FRONT  ( NT,LT,1, 0 )
      CALL EXPAND ( LSUB       )

  RETURN
  END
  SUBROUTINE FRONT ( NT,LT,NG,KZ )
  COMMON           NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
  COMMON           NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
  COMMON           NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
  COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)

      IF ( NG.GT.0 )                                GOTO 2
      CALL UNIT1 ( LT,2,NODE2,IQ2 )

  DO 1 I   = 1,NODE2
      K    = IQ2(I)
      IF ( NDOF(K).GT.0 )  IC(K) = IC(K) - 1
      IF ( NDOF(K).LT.0 )  IC(K) = IC(K) + 1
1     CONTINUE

2     N2= 0
  DO 3 I = 1,NOPT
      IF ( IB(I).EQ.IC(I) )                         GOTO 3
      IF ( IC(I).EQ.0     )  IB(I)=0
      N2      = N2 + 1
      LOC(N2) = I
3     CONTINUE

      IF ( N2.EQ.0 )                                GOTO 6

      M2 = 0
  DO 4 I = 1,N2
      MOC2(I)=0
      J = LOC(I)
      IF ( IC(J).NE.0 )                             GOTO 4
      LOC(I)=0
      M2 = M2 + 1
      MOC2(M2) = J
4     CONTINUE
```

```
      K2 = M2 + 1

      CALL GAPS ( NT,KZ )

 6 RETURN
   END
   SUBROUTINE GAPS ( NT,KZ )
   COMMON            NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
   COMMON            NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
   COMMON            NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
   COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT

      IF ( KZ .EQ.0     )              GOTO 8
      IF ( NQ1.GT.MAXF )               GOTO 8
      IF ( NQ2.GT.MAXF )               GOTO 8

 1    I1 = MAX0 ( K1,K2 )
      I2 = MIN0 ( N1,N2 )
      IF (I1.GT. I2)                   GOTO 10

   DO 4 I=I1,I2

      K=MOC1(I)

   DO 2 J=1,N2
      IF (  LOC(J).EQ.K        )       GOTO 3
 2    CONTINUE
                                       GOTO 4
 3    MOC2(I)=K
      LOC(J)=0
 4    CONTINUE

   DO 7 I=1,N2
      IF ( LOC(I).EQ.0 )               GOTO 7
   DO 5 J=K2,N2
      IF ( MOC2(J).EQ.0        )       GOTO 6
 5    CONTINUE
 6    MOC2(J) = LOC(I)
      LOC (I)=0
 7    CONTINUE
                                       GOTO 10

 8    J=K2
   DO 9 I=1,N2
      IF ( LOC(I).EQ.0 )               GOTO 9
      MOC2(J)=LOC(I)
      LOC(I)=0
      J = J + 1
 9    CONTINUE

10    NQ2 = NUM  ( N2,MOC2,NDOF )
      NT  = MAX0 ( NT,NQ2       )

   RETURN
   END
   SUBROUTINE UPDATE
   COMMON            NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
   COMMON            NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
   COMMON            NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)

   DO 1      I  = 1,NODE2
 1    IQ1 (I) = IQ2 (I)
```

```
      DO 3       I  = 1,N2
    3 MOC1(I) = MOC2(I)

      NODE1    = NODE2

      M1       = M2
      K1       = K2
      N1       = N2

      NQ1      = NQ2

      RETURN
      END
      SUBROUTINE EXPAND ( LZ )
      COMMON            NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
      COMMON            NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
      COMMON            NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
      COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)

      DO 1 I = 1,NODE1
      DO 1 J = 1,N1
    1 IF (  IQ1(I),EQ,MOC1(J) )  LQ(I) = J

      LD = 0
      DO 2 I  = 1,NODE1
      LP = IQ1 ( I )
      J2 = NUM ( LQ(I),MOC1,NDOF )
      J1 = J2 - IABS(NDOF(LP)) + 1
      DO 2 J  = J1,J2
      LD = LD + 1
    2 LQT(LD)=J

      MD1 = NUM ( NODE1,IQ1,NDOF )
      MQ  = NUM ( M1,MOC1,NDOF )
      KQ  = MQ+1
      NQ  = NQ1

      IF ( K1,GT,N1 )                          GOTO 8

      DO 5 I = K1,N1
      DO 3 J = 1, N2
      IF ( MOC1(I),EQ,MOC2(J) )                GOTO 4
    3 CONTINUE
    4 LOC ( I ) = J
    5 CONTINUE

      IF ( MQ,EQ,0 )                           GOTO 9

      DO 6 I = 1,MQ
          LD  = LD + 1
    6 LQT(LD) = 0

    9 DO 7 I=K1,N1
      LP = MOC1 ( I )
      J2 = NUM ( LOC(I),MOC2,NDOF )
      J1 = J2 - IABS(NDOF(LP)) + 1
      DO 7 J=J1,J2
      LD=LD+1
    7 LQT(LD)=J

    8 NQR = NQ2
```

```
      CALL  WRIT3 (2)
      NQL = NQ

      CALL PRNT0 ( LZ )
      CALL PRNT1
      CALL PRNT2 ( LZ )

   RETURN
   END
   FUNCTION NUM ( IPOS,MOC,NDOF )
   DIMENSION MOC(1),NDOF(1)

      NUM = 0
      IF ( IPOS.EQ.0 )                                    GOTO 2

   DO 1 I = 1,IPOS
      K = MOC ( I )
1     NUM = NUM + IABS ( NDOF ( K ) )

2  RETURN
   END
   SUBROUTINE FRONTST
   COMMON              B(700),A(13000)
   COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
   COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
   COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT
   COMMON / WRIT /   LPU4,NBAC,NW1,NW2,NC64,NOLD

      LPU4 = 0
      NOLD = 0

      IM2  = MSUB + 1

      NOEL = ISAV
   DO 3 IM = 1,IM2
3     CALL SUBSTR  ( NT,IM )

   DO 4 IM = 1,6
4     IF ( IM.NE.4 )  CALL WIND ( IM )

   RETURN
   END
   SUBROUTINE SUBSTR ( NT,IM )
   COMMON              B(700),A(13000)
   COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
   COMMON / SUBT /   NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
   COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT
   DATA L1,L2/1,2/

      IPOS = 0

   DO 1 I = 1, 700
1     B(I) = 0
   DO 2 I = 1,13000
2     A(I) = 0

      IF ( IM.LE.MSUB )                                   GOTO 3

      CALL SCANIQ ( NOEL,L1,L2,IPOS )
                                                          GOTO 5
3     CALL WIND ( L1 )
```

```
      CALL READ3 (1)

      NUEL = 0
   DO 4 IT = 1,NSUB
      LSUB = ISUB ( IT )
      CALL SCANIQ ( LSUB,L1,L2,IPOS )
      NUEL = NUEL + LSUB
      IF ( NQ,LE,MAXF )  CALL SETDISK (  L1,IPOS )
4     CONTINUE

      CALL WRIT4 (1,NE)

      CALL MERGEST ( L1,L2,IPOS )

5     IF(IPRNT,GE,2) PRINT 20,IPOS

20 FORMAT ( 6X *IPOS IS * I4 )
   RETURN
   END
   SUBROUTINE MERGEST ( L1,L2,IPOS )
   COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
   COMMON / SCR1 /  IR,LP,I1,I2,NR,MD,NW,S(2041)

      NOEL = NOEL - NUEL
      IF ( NOEL,EQ,0)                                  GOTO 3

   DO 2 I = 1,NOEL

1     CALL SEMB1 ( L2,2,IPOS )
      CALL SEMB1 ( L1,1,IPOS )

      IF ( I2,LT,MD )                                  GOTO 1

2     CONTINUE

3     NOEL = NOEL + NSUB

      CALL WIND ( L1 )
      CALL WIND ( L2 )

      LT  =  L1
      L1  =  L2
      L2  =  LT

   RETURN
   END
   SUBROUTINE SCANIQ ( LSUB,L1,L2,IPOS )
   COMMON            B(700),A(13000)
   COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
   COMMON / 1111 /   R1,R2,RM,WM,H1,H2,NE,NC
   COMMON / 0000 /   MAXF,MAXB,MAXW,IPRNT
   INTEGER           R1,R2,RM,WM,H1,H2
   COMMON / 2222 /   LDO,LOLD(700)
   LOGICAL     TESTLF,TESTRF,DISKLT,DISKEQ,SETCOR
   DATA  L5 , L6  /  5 , 6  /

                          LDO    = 0
   DO 2  LZ = 1,LSUB
                          CALL READ3 (2)
                          CALL FORMNO ( NQ )

      TESTLF = (NQL,GT,MAXF  ,OR,  NQ  ,GT,MAXF)  ,AND,  LZ,GT,1
```

```
      TESTRF = (NQ .GT.MAXF .OR. NQR.GT.MAXF)

      DISKLT =       TESTRF .AND. LZ .LT. LSUB
      DISKEQ =       TESTRF .AND. LZ .EQ. LSUB
      SETCOR = .NOT. TESTRF .AND. LZ .LT. LSUB

                     JPOS  = 0
                     NB    = 0

1                    NB    = NB + 1
                     CALL ROWS    ( NB )

      CALL PRNTS(1,LZ,NB,NOTH,NOTH,LP)

                     CALL SEMBLE1 ( NB,L2 )
      IF ( TESTLF  ) CALL SEMBLE2 ( NB,L5 )

                     CALL PRNTA   (A,B,R2,NQ)
      IF ( NB.EQ.1 )       R1 = 2
      IF ( MQ.GT.0)  CALL REDUCE  ( NB,MQ,NQ )
      IF ( NB.EQ.1 )       R1 = KQ
      IF ( DISKLT  ) CALL SETDISK ( L6,JPOS )
      IF ( DISKEQ  ) CALL SETDISK ( L1,IPOS )

      IF ( R2.LT.NQ )                                        GOTO 1

                     CALL WRIT4   (2,NE)
                     CALL ZEROA   (NE,L5,L6,TESTLF,TESTRF )

      IF ( SETCOR  ) CALL SETCORE ( NQ,NQR )

      IF ( TESTRF  ) CALL PRNTS   ( 2,LZ,NOTH,JPOS,NOTH,LP )

2     CONTINUE

  RETURN
  END
  SUBROUTINE ZEROA (NE,L5,L6,TESTLF,TESTRF)
  COMMON           B(700),A(13000)
  COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
  COMMON / 2222 /  LDO,LOLD(700)
  LOGICAL TESTLF,TESTRF

      IF ( MQ.EQ.0 )                                 GOTO 3

  DO 1 I = 1,MQ
1    B(I) = 0

  DO 2 I = 1,NE
2    A(I) = 0

3     LDO  = 0

      IF (.NOT.TESTRF)                               GOTO 5

      L = 0
  DO 4 I = KQ,NQ
      L = L + 1
4     LOLD (L) = LQT (I+MD1)
      LDO = L

      LT  =  L5
```

```
      L5  =  L6
      L6  =  LT

5     IF ( TESTLF.OR.TESTRF )  CALL WIND ( L5 )
      IF ( TESTLF.OR.TESTRF )  CALL WIND ( L6 )

  RETURN
  END
  SUBROUTINE ROWS ( NB )
  COMMON / NOOO /  NO(700)
  COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
  COMMON / 0000 /  MAXF,MAXB,MAXW,IPRNT
  COMMON / 1111 /  R1,R2,RM,WM,H1,H2,NE,NC
  INTEGER          R1,R2,RM,WM,H1,H2
  COMMON / 2222 /  LDO,LOLD(700)

      IF ( NB.GT.1 )                                  GOTO  1
      R1 =      1
      R2 = MQ + 1
      RM = 0
      WM = 0
      NE = 0
      IF ( MQ.GT.0 )  NE = NO(MQ) + NQ - MQ
      IF ( NO(NQ).GT.MAXW )                           GOTO  2
      R2 = NQ
      NC = NO(NQ) - NE

                                                      GOTO  6
1     R1 = R2 +  1
      R2 = R1
2     NC = NE
      IT = NQ - MQ - RM
      I1 = R2
  DO  3  I  = I1,NQ
      NC = NC + IT
      IF ( NC.GT.MAXW )                               GOTO  4
      R2 = R2 +  1
3     IT = IT -  1

                                                      GOTO  5
4     NC = NC - IT
5     R2 = R2 -  1

      NC = NC - NE

6     H1 = 0
      H2 = 0
  DO  8  I  = R1,R2
  DO  7  J  =  1,MD1
      IF ( LQT (J).NE.I )                             GOTO  7
      H1 = H1 + 1

                                                      GOTO  8
7     CONTINUE
8     CONTINUE

      IF ( LDO.EQ.0 )                                 GOTO 11

  DO 10  I  = R1,R2
  DO  9  J  =  1,LDO
      IF ( LOLD(J).NE.I )                             GOTO  9
      H2 = H2 + 1

                                                      GOTO 10
 9    CONTINUE
10    CONTINUE
```

```
11 RETURN
   END
   SUBROUTINE SEMBLE1 ( NB,L2 )
   COMMON            B(700),A(13000)
   COMMON / NOOO /   NO(700)
   COMMON / FRNT /   LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
   COMMON / SCR1 /   IR,LP,I1,I2,NR,MD,NW,S(2041)
   COMMON / 1111 /   R1,R2,RM,WM,H1,H2,NE,NC
   INTEGER           R1,R2,RM,WM,H1,H2,R    ,BACK,FINI

        IF ( H1   ,EQ,0 )                                  GOTO 5


        IF ( NB   ,EQ,1 )          LOGO = 0
                                   LOGR = 0
                                   BACK = 0
                                   FINI = 0
                                   IDEC = 0

 1 IF (LOGO,NE,1) CALL SEMB1 (L2,2,NOTH)
        LOGO =  0

   CALL PRNTS(3,NOTH,NOTH,NOTH,IDEC,LP)

        R = 0
        L = NR
   DO 3 M = I1,I2
        R = R + 1
        I = LQT(M)
        K = LQT(M) - RM
        IF ( I,GT,R2 )             FINI = 1
        IF ( I,LT,R1 )                                     GOTO 2
        IF ( I,GT,R2 )                                     GOTO 2
                                   H1   = H1   - 1
                                   B(K) = B(K) + S(R)

 2 DO 3 N = M , MD
        I = MIN0(LQT(M),LQT(N))
        J = MAX0(LQT(M),LQT(N))
        K = NO(I) - I + J - WM
        L = L + 1
        IF ( I,LT,R1 )                                     GOTO 3
        IF ( I,GT,R2 )                                     GOTO 3
                                   A(K) = A(K) + S(L)
 3        CONTINUE

          IF ( BACK+FINI,EQ,0 )                            GOTO 4
                                   LOGR = LOGR + 1
                                   BACK = BACK + IR

 4        IF ( H1   ,GT,0 )                                GOTO 1

   CALL PRNTS(4,NOTH,NOTH,NOTH,NOTH,LP)

                                   LOGO = LOGR
   IF (LOGO,GT,1) CALL SEMB1 (L2,3,BACK)

 5 RETURN
   END
   SUBROUTINE SEMBLE2 ( NB,L5 )
   COMMON            B(700),A(13000)
```

```
      COMMON / NOOO /  NO(700)
      COMMON / 2222 /  LDO,LOLD(700)
      COMMON / SCR2 /  IR,LP,I1,I2,NR,MD,NW,S(2041)
      COMMON / 1111 /  R1,R2,RM,WM,H1,H2,NE,NC
      INTEGER          R1,R2,RM,WM,H1,H2,R     ,BACK,FINI

           IF ( H2  .EQ.0 )                                  GOTO 5


           IF ( NB  .EQ.1 )          LOGO = 0
                                     LOGR = 0
                                     BACK = 0
                                     FINI = 0
                                     IDEC = 0

    1 IF (LOGO.NE.1) CALL SEMB2 (L5,2,NOTH)
           LOGO =  0

      CALL PRNTS(3,NOTH,NOTH,NOTH,IDEC,LP)

           R = 0
           L = NR
      DO 3 M = I1,I2
           R = R + 1
           I = LOLD(M)
           K = LOLD(M) - RM
           IF ( I.GT.R2 )          FINI = 1
           IF ( I.LT.R1 )                                    GOTO 2
           IF ( I.GT.R2 )                                    GOTO 2
                                   H2   = H2   - 1
                                   B(K) = B(K) + S(R)

    2 DO 3 N = M , MD
           I = MIN0(LOLD(M),LOLD(N))
           J = MAX0(LOLD(M),LOLD(N))
           K = NO(I) - I + J - WM
           L = L + 1
           IF ( I.LT.R1 )                                    GOTO 3
           IF ( I.GT.R2 )                                    GOTO 3
                                   A(K) = A(K) + S(L)
    3        CONTINUE

             IF ( BACK+FINI.EQ.0 )                           GOTO 4
                                   LOGR = LOGR + 1
                                   BACK = BACK + IR

    4        IF ( H2  .GT.0 )                                GOTO 1

      CALL PRNTS(4,NOTH,NOTH,NOTH,NOTH,LP)

                                   LOGO = LOGR
      IF (LOGO.GT.1) CALL SEMB2 (L5,3,BACK)

    5 RETURN
      END
      SUBROUTINE REDUCE ( NB,MQ,NQ )
      COMMON            B(700),A(13000)
      COMMON / NOOO /   NO(700)
      COMMON / 1111 /   R1,R2,RM,WM,H1,H2,NE,NC
      INTEGER           R1,R2,RM,WM,H1,H2
```

```
      DO 1 I  = R1,R2

           L  =          I - RM
           IO = NO(I) - I - WM
           J2 = I       - 1
           IF ( I,GT,MQ ) J2 = MQ

      DO 1 J  = 1,J2

           JO = NO(J) - J
           C  =-A(JO+I)/A(JO+J)
           B(L) = B(L) + B(J) * C

      DO 1 K  = I,NQ
    1      A(IO+K) = A(IO+K) + A(JO+K)*C

      RETURN
      END
      SUBROUTINE SETDISK ( L6,LPOS )
      COMMON             B(700),A(13000)
      COMMON / NOOO /    NO(700)
      COMMON / FRNT /    LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / STOR /    IR,LP,J1,J2,NR,MD,NW
      COMMON / 1111 /    R1,R2,RM,WM,H1,H2,NE,NC
      COMMON / 0000 /    MAXF,MAXB,MAXW,IPRNT
      INTEGER            R1,R2,RM,WM,H1,H2

           MD = NQ - MQ

           I2 = R1 - 1
           IT = NQ - MQ - RM + 1

    1      I1 = I2 + 1
           I2 = I1
           NW =  7
      DO   2  I  = I1 , R2
           NW = NW + IT
           IF (NW,GT,MAXB)                          GOTO  3
           I2 = I2 +  1
    2      IT = IT -  1
                                                    GOTO  4
    3      NW = NW - IT
    4      I2 = I2 - 1
           NR = I2 - I1 + 1

           J1 = I1                  - MQ
           J2 = I2                  - MQ
           K1 = I1                  - RM
           K2 = I2                  - RM
           L1 = NO(I1)              - WM
           L2 = NO(I2) + NQ - I2    - WM

           CALL STORE ( L6,LPOS,K1,L1,B,A )

      DO   5  I = K1,K2
    5       B(I)=0

      DO   6  I = L1,L2
    6       A(I)=0

           CALL PRNTS(5,NOTH,NOTH,NOTH,NOTH,LP)
```

```
          IF (I2,LT,R2)                                    GOTO  1

          RM = RM + R2 - R1 +  1
          WM = WM + NC

    RETURN
    END
    SUBROUTINE SETCORE ( NQ,NR )
    COMMON             B(700),A(13000)
    COMMON / NOOO /    NO(700)

          IF ( NR,EQ,0 )                                   GOTO 7
          IF ( NR - NQ )                                    1,6,3

1         CALL SCRMBLE ( NQ )

          L  = NR
    DO 2  I  = 2,NR
          N  = NO(I) - I
    DO 2  J  = I,NR
          L  =  L + 1
          A(L) = A(N+J)
2         A(N+J) = 0

          CALL FORMNO ( NR,NO )
                                                           GOTO 7
3         IN = NR - NQ
          L2 = (NR*(NR+1))/2 - (IN*(IN+1))/2 + 1

    DO 5  II = 1,NQ
          I  = NQ - II + 1
          IN = IN + 1
          L1 = NO(I) + II
          L2 = L2 - IN + II
    DO 4  L  = 1,II
          TEMP      = A (L1-L)
                      A (L1-L) = 0
4         A (L2-L) = TEMP
5         L2 = L2 - II

          CALL FORMNO ( NR,NO )

6         CALL SCRMBLE ( NR )

7         RETURN
          END
    SUBROUTINE SCRMBLE ( NT )
    COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)

         KKOUNT = 0

          IF (NQ .GE. NT)                                  GOTO 2

          J = NQ + 1
    DO 1  I = J,NT
1         LQT (I+MD1) = 0

2   DO 4  M = KQ,NQ

3         N = LQT(M+MD1)

          IF ( N .EQ. M .OR. N .EQ. 0)                     GOTO 4
```

```
      LQT(M+MD1) = LQT(N+MD1)
      LQT(N+MD1) =      N

      I          = MIN0(M,N)
      J          = MAX0(M,N)

      CALL SWITCH (I,J,NT)

      KKOUNT  = KKOUNT + 1
      IF (KKOUNT .GT. NT) STOP
                                                    GOTO 3
4 CONTINUE

  RETURN
  END
  SUBROUTINE SWITCH ( M,N,NT )
  COMMON            B(700),A(13000)
  COMMON / NOOO /   NO(700)


      C    =B(M)
      B(M)=B(N)
      B(N)=C


      MS=NO(M)-M
      NS=NO(N)-N


      C       =A(MS+M)
      A(MS+M)=A(NS+N)
      A(NS+N)=C


      IF (  M.EQ.1              )  GOTO 2
      I2=M-1
  DO 1 I= 1,I2
      IS=NO(I)-I
      C       =A(IS+M)
      A(IS+M)=A(IS+N)
1     A(IS+N)=C


2     J1=M+1
      IF ( J1.EQ.N              )  GOTO 4
      J2=N-1
  DO 3 J=J1,J2
      JS=NO(J)-J
      C       =A(MS+J)
      A(MS+J)=A(JS+N)
3     A(JS+N)=C


4     IF (  N.EQ.NT             )  GOTO 6
      J1=N+1
  DO 5 J=J1,NT
      C       =A(MS+J)
      A(MS+J)=A(NS+J)
5     A(NS+J)=C

6 RETURN
```

```
      END
      SUBROUTINE BACKSUB
      COMMON           B(700),A(13000)
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      DATA  L5 , L6  /  5 , 6  /

           CALL READ4 (3)
           IF ( MSUB.GT.0 )                          GOTO 1
           CALL SOLVE (  1,    1,NOEL,L6 )
                                                     GOTO 4
1          LEFT = NOEL - 1
           CALL SOLVE (  1,    1,LEFT,L6 )
           CALL SOLVE (  1,    2,   1,L6 )

      DO 3 IM = 1,MSUB

           CALL WIND ( L5 )
           CALL WIND ( L6 )
           LT  =  L5
           L5  =  L6
           L6  =  LT

           CALL READ4 (1)

      DO 2 IT = 1,NSUB
           LT   = NSUB - IT + 1
           LSUB = ISUB ( LT )
           CALL UNITED (L5,2,MD,B)
           CALL PRNTB ( B,MD )
2          CALL SOLVE ( IM,MSUB,LSUB,L6 )

3          CONTINUE

4     RETURN
      END
      SUBROUTINE SOLVE ( IM,JM,LSUB,L6 )
      COMMON           B(700),A(13000)
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / SCR1 /  IR,LP,I1,I2,NR,MD,NW,S(2041)
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)

           IF ( LSUB.EQ.0 )                          GOTO 8

      DO 7 LZ = 1,LSUB

           CALL READ4 (2)
           CALL PRNT2 ( LZ )

           CALL FORMNQ ( NQ )

           IF ( KQ.GT.NQ )                           GOTO 2

      DO 1 I  = KQ,NQ
           L       = LQT ( I + MD1 )
1          S ( I ) =   B ( L )

2     DO 3 I  =  1,NQ
3          B ( I ) =   S ( I )

           CALL PRNTB ( B,NQ )
           CALL PRNTA ( A,B,MQ,NQ )
           IF ( MQ.GT. 0 )  CALL BPASS ( MQ,KQ,NQ )
```

```
        CALL PRNTB ( B,NQ )

  4 DO 5 I  = 1,MD1
        L         = LQT ( I )
  5     S  ( I ) =   B ( L )

        IF ( IM,NE,JM )                                    GOTO 6

        CALL PRNTED ( 2,ISAV,MD1,S )
        CALL UNITED ( 1,   1,MD1,S )
                                                           GOTO 7
  6     CALL PRNTED ( 1,NSTR,MD1,S )
        CALL UNITED (L6,   1,MD1,S )

  7 CONTINUE

  8 RETURN
    END
    SUBROUTINE BPASS ( MQ,KQ,NQ )
    COMMON           B(700),A(13000)
    COMMON / NOOO /  NO(700)

        IF ( MQ ,EQ, NQ          )                         GOTO 2

    DO 1 I = 1,MQ
        K = NO(I)-I
    DO 1 J = KQ,NQ
  1     B(I) = B(I)-A(J+K)*B(J)

  2     M = MQ+1
        I = NO(MQ)
        B(MQ) = B(MQ)/A(I)

        IF ( MQ,EQ,1 )                                     GOTO 5

    DO 4 L = 2,MQ

        I = M-L
        J1 = I+1
        K = NO(I)-I

    DO 3 J = J1,MQ
  3     B(I) = B(I)-A(J+K)*B(J)

        K = NO(I)

  4     B(I) = B(I)/A(K)

  5 RETURN
    END
    SUBROUTINE FORMNO ( NT )
    COMMON / NOOO /  NO(700)

        NO(1) = 1
        L = NT+2
    DO 1 I = 2,NT
  1     NO(I) = NO(I-1)+L - I

    RETURN
    END
    SUBROUTINE UNIT1 ( NTAPE,IGO,NODES,IQ )
    DIMENSION IQ(1)
```

```
      COMMON / TRAC /  NWS,NWR,NBS,NUP
                                                      GOTO (1,2) IGO
    1 CALL IOP (2HWB,NTAPE,NODES,     1)
      CALL IOP (2HWB,NTAPE,IQ    ,NODES)
      NWS = NWS + NODES + 1
                                                      GOTO 3
    2 CALL IOP (2HRB,NTAPE,NODES,     1)
      CALL IOP (2HRB,NTAPE,IQ    ,NODES)
      NWR = NWR + NODES + 1

    3 RETURN
      END
      SUBROUTINE WRIT3 ( IGO )
      COMMON / TRAC /  NWS,NWR,NBS,NUP
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      DIMENSION        IDUM(956)
      EQUIVALENCE ( MQ,IDUM )
                                                     GOTO ( 1,2 ) IGO
    1 CALL IOP (2HWB,3,NSUB,64)
      NWS = NWS + 64
                                                      GOTO 3
    2           LD = LD + 6
      CALL IOP (2HWB,3,LD ,  1)
      CALL IOP (2HWB,3,IDUM,LD)
      NWS = NWS + LD + 1
      NUP = NUP + (MQ*(MQ*MQ+3*NQ*NQ-3*MQ*NQ-1))/6

    3 RETURN
      END
      SUBROUTINE READ3 ( IGO )
      COMMON / TRAC /  NWS,NWR,NBS,NUP
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      DIMENSION        IDUM(956)
      EQUIVALENCE ( MQ,IDUM )
                                                     GOTO ( 1,2 ) IGO
    1 CALL IOP (2HRB,3,NSUB,64)
      NWR = NWR + 64
                                                      GOTO 3
    2 CALL IOP (2HRB,3,LD ,  1)
      CALL IOP (2HRB,3,IDUM,LD)
      NWR = NWR + LD + 1

    3 RETURN
      END
      SUBROUTINE WRIT4 ( IGO,NE )
      COMMON           B(700),A(13000)
      COMMON / TRAC /  NWS,NWR,NBS,NUP
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / SUBT /  NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
      COMMON / WRIT /  LPU4,NBAC,NW1,NW2,NC64,NOLD
      DIMENSION        IDUM(956)
      EQUIVALENCE ( MQ,IDUM )
                                                        GOTO ( 1,2 ) IGO
    1                IR  =1
                     LPU4=IR+LPU4
                     NBAC=IR+NOLD
                                 CALL IOP(2HWB,4,LPU4,   2)
                                 CALL IOP(2HWB,4,NSUB,  62)
                                 NWS = NWS + 64
                                                          GOTO 3
```

```
  2 NW =LD+MQ+NE+4
                      IR   =NW / 64
                      IT   =IR * 64
    IF( IT.NE.NW )    IR   =IR +  1
                      NW   =IR * 64
                      NW1  =LD
    IF( MQ.EQ. 0 )    NW1  =NW -  4
    NW2=NW-NW1-MQ-4
                      LPU4=IR+LPU4
                      NBAC=IR+NOLD
    IF(NW-NW1-NW2-MQ-4 .NE. 0) PRINT 100
100 FORMAT( 6X *ERROR IN UNIT4* )
                                    CALL IOP(2HWB,4,LPU4,    4)
                                    CALL IOP(2HWB,4,IDUM, NW1)
    IF( MQ.GT. 0 )                  CALL IOP(2HWB,4,B    ,  MQ)
    IF( MQ.GT. 0 )                  CALL IOP(2HWB,4,A    , NW2)
                                    NWS = NWS + NW
  3                 NC64=IR
                      NOLD=IR

    RETURN
    END
    SUBROUTINE READ4 ( IGO )
    COMMON             B(700),A(13000)
    COMMON / TRAC /    NWS,NWR,NBS,NUP
    COMMON / SCR1 /    IR,LP,I1,I2,NR,MD,NW,S(2041)
    COMMON / FRNT /    LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
    COMMON / SUBT /    NOEL,NOPT,ISAV,NSTR,MSUB,NUEL,NSUB,ISUB(63)
    COMMON / WRIT /    LPU4,NBAC,NW1,NW2,NC64,NOLD
    DIMENSION          IDUM(956)
    EQUIVALENCE ( MQ,IDUM )

  1                                 GOTO ( 1,2,4 )  IGO
                                    CALL IOP(2HRB,4,LPU4,    2)
                                    CALL IOP(2HRB,4,NSUB,   62)
                                    NWR = NWR + 64

  2                                                 GOTO 3
                                    CALL IOP(2HRB,4,LPU4,    4)
                                    CALL IOP(2HRB,4,IDUM, NW1)
    IF( MQ.GT. 0)                   CALL IOP(2HRB,4,S    ,  MQ)
    IF( MQ.GT. 0)                   CALL IOP(2HRB,4,A    , NW2)
    NW = NW1+NW2+MQ+4

                                    NWR = NWR + NW
  3                                 CALL IOP(2HSP,4,LPU4-NBAC)
                                    NBS = NBS + NBAC

  4                                                 GOTO 5
                                    CALL IOP(2HSP,4,LPU4-NC64)
                                    NBS = NBS + NC64

  5 RETURN
    END
    SUBROUTINE STORE ( NTAPE,LPOS,K1,L1,B,A )
    DIMENSION B(1),A(1)
    COMMON / STOR /  IR,LP,J1,J2,NR,MD,NW
    COMMON / TRAC /  NWS,NWR,NBS,NUP

                      IR   = NW / 64
                      IT   = IR * 64
    IF( IT.NE.NW )    IR     IR +  1
                      NW   = IR *64 - 7
                      NA   = NW - NR
                      LPOS = IR + LPOS
                      LP   =      LPOS
```

```
                    CALL IOP(2HWR,NTAPE,IR,7)
                    CALL IOP(2HWB,NTAPE,B(K1),NR)
                    CALL IOP(2HWB,NTAPE,A(L1),NA)
                    NWS = NWS + NW + 7

      RETURN
      END
      SUBROUTINE SEMB1 ( NTAPE,IGO,LPOS )
      COMMON / TRAC /  NWS,NWR,NBS,NUP
      COMMON / SCR1 /  IR,LP,I1,I2,NR,MD,NW,S(2041)

                                  GOTO ( 1,2,3 )    IGO
 1                  LPOS = IR +     LPOS
                    LP   =      LPOS

                    CALL IOP(2HWB,NTAPE,IR ,  7)
                    CALL IOP(2HWR,NTAPE,S  , NW)
                    NWS = NWS + NW + 7
                                                    GOTO 4
 2                  CALL IOP(2HRB,NTAPE,IR ,  7)
                    CALL IOP(2HRB,NTAPE,S  , NW)
                    NWR = NWR + NW + 7
                                                    GOTO 4
 3                  CALL IOP(2HSP,NTAPE,LP-LPOS)
                    NBS = NBS + LPOS

 4 RETURN
      END
      SUBROUTINE SEMB2 ( NTAPE,IGO,LPOS )
      COMMON / TRAC /  NWS,NWR,NBS,NUP
      COMMON / SCR2 /  IR,LP,I1,I2,NR,MD,NW,S(2041)

                                  GOTO ( 1,2,3 )    IGO


 1                  CALL IOP(2HWB,NTAPE,IR , 7)
                    CALL IOP(2HWB,NTAPE,S  , NW)
                    NWS = NWS + NW + 7
                                                    GOTO 4
 2                  CALL IOP(2HRB,NTAPE,IR ,  7)
                    CALL IOP(2HRB,NTAPE,S  , NW)
                    NWR = NWR + NW + 7
                                                    GOTO 4
 3                  CALL IOP(2HSP,NTAPE,LP-LPOS)
                    NBS = NBS + LPOS

 4 RETURN
      END
      SUBROUTINE UNITED (NTAPE,IGO,MD,B)
      DIMENSION B(1)
      COMMON / TRAC /  NWS,NWR,NBS,NUP
                                              GOTO (1,2) IGO

 1 CALL IOP (2HWB,NTAPE,MD, 1)
      CALL IOP (2HWR,NTAPE,B ,MD)
      NWS = NWS + MD + 1
                                              GOTO 3

 2 CALL IOP (2HRB,NTAPE,MS, 1)
      CALL IOP (2HRB,NTAPE,B ,MS)
      NWR = NWR + MS + 1

 3 RETURN
```

```
      END
      SUBROUTINE WIND ( NTAPE )

      CALL IOP (3HREW,NTAPE)

      RETURN
      END
      SUBROUTINE PRNTED ( JGO,NUMB,MD1,ED )
      COMMON / 0000 /  MAXF,MAXB,MAXW,IPRNT
      DIMENSION ED(1)

      IF (IPRNT.EQ.0)                          GOTO 1
      IF ( JGO.EQ.1 )  PRINT 100,NUMB
      IF ( JGO.EQ.2 )  PRINT 200,NUMB

      PRINT 300,(ED (I),I=1,MD1)

      NUMB = NUMB + 1

  100 FORMAT (       6X *SUBSTR * I5 )
  200 FORMAT (       6X *ELEMENT* I5 )
  300 FORMAT (      18X 12F5.1         )
    1 RETURN
      END
      SUBROUTINE PRNT0 ( LZ )
      COMMON             NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
      COMMON             NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
      COMMON             NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      NUM1 = NUM1 + 1
      IF ( NUM1.LE.NUM2 )  GOTO 99
      IF (IP0 .EQ. 0) GOTO 99
      PRINT 10,LZ,NODE1,MD1
      PRINT 20,M1,K1,N1
      PRINT 30,MQ,KQ,NQ
    5 FORMAT ( *1* )
   10 FORMAT ( // 6X *LZ    =* I4,6X *NODE1=* I4,6X *MD1   =* I4 )
   20 FORMAT (    6X *M1    =* I4,6X *K1    =* I4,6X *N1    =* I4 )
   30 FORMAT (    6X *MQ    =* I4,6X *KQ    =* I4,6X *NQ    =* I4 )
   99 RETURN
      END
      SUBROUTINE PRNT1
      COMMON             NDOF(2500),IB(2500),IC(2500),LQ(150),LOC(150)
      COMMON             NODE1,IQ1(150),M1,K1,N1,NQ1,MOC1(150)
      COMMON             NODE2,IQ2(150),M2,K2,N2,NQ2,MOC2(150)
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      IF ( NUM1.LE.NUM2 )  GOTO 99
      IF (IP1 .EQ. 0) GOTO 99
      PRINT 10,(IQ1 (I),I=1,NODE1)
      PRINT 20,(LQ  (I),I=1,NODE1)
      PRINT 40,(MOC1(I),I=1,N1   )
      PRINT 50,(MOC2(I),I=1,N2   )
      PRINT 60,(LOC (I),I=1,N1   )
   10 FORMAT (// 6X *IQ  *  20I4 )
   20 FORMAT (    6X *LQ  *  20I4 )
   40 FORMAT (    6X *MOC1*  20I4 )
   50 FORMAT (    6X *MOC2*  20I4 )
   60 FORMAT (    6X *LOC *  20I4 )
   99 RETURN
      END
```

```
      SUBROUTINE PRNT2 ( LZ )
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      IF ( NUM1.LE.NUM2 )  GOTO 99
      IF (IP2 .EQ. 0) GOTO 99
      PRINT 1,LZ
      PRINT 10,(LQT (I),I=1,MD1)
      I1 = MD1 +  1
      I2 = MD1 + NQ
      PRINT 11,(LQT(I),I=I1,I2 )
    1 FORMAT (   6X *LZ  *    I4 )
   10 FORMAT (   6X *LQT * 20I4 )
   11 FORMAT (   6X *LQCT* 20I4 )
   99 RETURN
      END
      SUBROUTINE PRNTA (A,B,NQ,NT)
      COMMON / NOOO /  NO(450)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      DIMENSION A(1),B(1),Q(108)
      IF (IPA .EQ. 0) GOTO 99
      PRINT 20
      DO 1 I=1,NT
    1 Q(I)=0
      PRINT 10,(A(I),I=1,NT),B(1)
      IF ( NQ.EQ.1 )  GOTO 99
      DO 2 K=2,NQ
      I1=K-1
      I2=NO(K)
      I3=I2+NT-K
    2 PRINT 10,(Q(I),I=1,I1),(A(I),I=I2,I3),B(K)
   10 FORMAT ( 6X *A   * 17F6.0 )
   20 FORMAT ( / )
   99 RETURN
      END
      SUBROUTINE PRNTB ( B,NQ )
      DIMENSION B(1)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      IF ( IPB.EQ.0 )  GOTO 99
      PRINT 100,(B(I),I=1,NQ)
  100 FORMAT ( /  6X *B   * 10F6.1 )
   99 RETURN
      END
      SUBROUTINE PRNTS(NGO,LZ,NB,JPOS,IDEC,LP)
      COMMON / PRNT /  IP0,IP1,IP2,IPA,IPB,NUM1,NUM2,IPS
      COMMON / FRNT /  LD,MQ,KQ,NQL,NQ,NQR,MD1,LQT(950)
      COMMON / STOR /  IR,LN,J1,J2,NR,MD,NW
      COMMON / 1111 /  R1,R2,RM,WM,H1,H2,NE,NC
      INTEGER          R1,R2,RM,WM,H1,H2

      IF ( IPS.EQ.0 )                                  GOTO 6
                                                       GOTO ( 1,2,3,4,5 )  NGO
    1 IF ( NB.EQ.1 ) PRINT 10,LZ,MQ,NQ,NE
                     PRINT 15,NB,R1,R2,NC
                                                       GOTO 6
    2 PRINT 30,LZ,JPOS
                                                       GOTO 6
    3 IDEC = IDEC +  1
      IF   ( IDEC.EQ.1 ) PRINT 100,LP
                                                       GOTO 6
    4 PRINT 200,LP
                                                       GOTO 6
    5 I1 = J1 + MQ
```

```
      I2 = J2 + MQ
      PRINT 300,I1,I2,IR,LN

   10 FORMAT ( / 12X *LZ* I4,3X *MQ* I4,3X *NQ* I4,3X *NE* I6 )
   15 FORMAT (   12X *NB* I4,3X *R1* I4,3X *R2* I4,3X *NC* I6 )
   30 FORMAT  ( 6X *LZ  *I3 , 3X *JPOS* I3 )
  100 FORMAT ( 12X *BEGINNING POSITION IS* I6 )
  200 FORMAT ( 12X *TERMINAL  POSITION IS* I6 )
  300 FORMAT ( 12X *I1*I4,3X*I2*I4,3X*IR*I4,3X*LP*I4 )

    6 RETURN
      END
```

APPENDIX C

FORTRAN LISTINGS OF

SKYLINE EQUATION SOLVERS

APPENDIX C.1

SKYSOL SKYLINE EQUATION SOLVER

```
      SUBROUTINE SKYFAC (A,NBEG,NEND,N,LD,V,SINGAB,
     . PDCHEK,ICOND,IAFILE,ACOND,DETCF,IDETEX,NEGEIG,IFAIL)
C
C     TYPE AND DIMENSION STATEMENTS
C
      REAL A(1),V(1)
      REAL AIJ2,D,DETCF,DOTPRD,EPSMAC
      INTEGER LD(1)
      LOGICAL PDCHEK,SINGAB
      EQUIVALENCE (AIJ2,D,DMAX)
      DATA EPSMAC/7.11E-15/
C
C     INITIALIZATION
C
      IFAIL=0
      IDETEX=0
      NEGEIG=0
C
C     SAVE ORIGINAL MATRIX IF IAFILE NE 0 AND NBEG=0
C
      NWA=IABS(LD(N+1))
      IF(IAFILE.EQ.0) GO TO 200
      IF(NBEG.NE.0) GO TO 200
      REWIND IAFILE
      WRITE(IAFILE) (A(J),J=1,NWA)
C
C     COMPUTE SQUARED LENGTHS OF UNCONSTRAINED ROWS NBEG+1 THRU N
C
  200 NBEGP1=NBEG+1
      DO 1000 I=NBEGP1,N
       II=LD(I+1)
       IF(II) 1000,1000,400
  400  V(I)=A(II)**2
       M=II-I
       K=MAX0(NBEGP1,IABS(LD(I))-M+1)
       L=MIN0(NEND,I)-1
       IF(K-L) 500,500,1000
  500 DO 800 J=K,L
       IF(LD(J+1)) 800,800,600
  600  AIJ2=A(M+J)**2
       V(I)=V(I)+AIJ2
       V(J)=V(J)+AIJ2
  800  CONTINUE
 1000 CONTINUE
C
C     FACTORIZATION SECTION
C
      DO 4000 J=NBEGP1,NEND
C
C     COMPUTE KU SUPERDIAGONAL ENTRIES OF J-TH COLUMN OF (U)
C     IF UNCONSTRAINED
C
       JJ=LD(J+1)
       IF(JJ) 4000,4000,1200
 1200  D=A(JJ)
       JMJ=IABS(LD(J))
       JK=JJ-JMJ
       KU=JK-1
       IF(KU.EQ.0) GO TO 2200
      DO 2000 K=1,KU
       I=J-JK+K
```

PAGE 132 INTENTIONALLY BLANK

```
          V(K)=0.0
          II=LD(I+1)
          IF(II) 2000,2000,1800
 1800     M=MIN0(II-IABS(LD(I)),K)-1
          IJ=JMJ+K
          V(K)=A(IJ)-DOTPRD(A(II-M),V(K-M),M)
          A(IJ)=V(K)*A(II)
 2000   CONTINUE
C
C       COMPUTE DIAGONAL ELEMENT D(J)
C
          D=D-DOTPRD(A(JMJ+1),V,KU)
C
C       SINGULARITY TEST
C
 2200     TOLROW=8.*EPSMAC*SQRT(V(J))
          IF(ABS(D).GT.TOLROW) GO TO 2500
          IF(SINGAB) GO TO 6000
          D=TOLROW
 2500     A(JJ)=1.0/D
C
C       UPDATE DETERMINANT IF DETCF IS NONZERO
C
 3000     IF(DETCF.EQ.0.0) GO TO 3500
          DETCF=DETCF*D
 3200     IF(ABS(DETCF).LT.1.0) GO TO 3400
          DETCF=DETCF*.0625
          IDETEX=IDETEX+4
          GO TO 3200
 3400     IF(ABS(DETCF).GT..0625) GO TO 3500
          DETCF=DETCF+16.
          IDETEX=IDETEX-4
          GO TO 3400
C
C       POSITIVE DEFINITNESS CHECK (IF PDCHEK=.TRUE.)
C
 3500     IF(D.GT.0.0) GO TO 4000
          NEGEIG=NEGEIG+1
          IF(PDCHEK) GO TO 6500
 4000   CONTINUE
C
C       SAVE FACTORIZATION IF IAFILE NE 0 AND N=NEND
C
          IF(IAFILE.EQ.0) GO TO 5000
          IF(NEND.NE.N) GO TO 5000
          WRITE(IAFILE) (A(J),J=1,NWA)
          IF(ICOND.EQ.0) GO TO 5000
C
C       MATRIX CONDITION ESTIMATION
C
          K=0
          DMAX=0.0
          DO 4200 I=1,ICOND
C
C       SET UP RANDOM RHS VECTOR IN V
C
           CALL SKYRDM(V,NWA,0,0,K)
C
C       SOLVE FOR V AND DO ONE CYCLE OF ITERATIVE REFINEMENT TO
C       OBTAIN 2-NORM RELATIVE SOLUTION ERROR IN DELTA
C
          CALL SKYSOL (A,N,LD,DOTPRD,0,-1,V,V(N+1),1,IAFILE,
```

```
     .    V(2*N+1),DELTA)
          DMAX=AMAX1(DELTA,DMAX)
 4200 CONTINUE
C
C     NOW ESTIMATE C(A) FROM LARGEST DELTA AND MACHINE PRECISION
C
      ACOND=DMAX/((1.0+DMAX)*EPSMAC)
 5000 RETURN
C
C     ERROR EXITS
C      SINGULAR MATRIX(SINGAB=.TRUE.)
 6000 IFAIL=J
      DETCF=0.0
      GO TO 5000
C     INDEFINITE MATRIX (PDCHEK=.TRUE.)
 6500 IFAIL=-J
      GO TO 5000
      END
      SUBROUTINE SKYSOL(A,N,LD,DOTPRD,IOP,IBX,B,X,
     . IREF,IAFILE,V,DELTA)
C
C     TUYPE AND DIMENSION STATEMENTS
C
      INTEGER LD(1)
      REAL A(1),B(1),BI,BXFAC,DELTA
      REAL RNORM,V(1),X(1),XI,XNORM
      EQUIVALENCE (BI,XI,XNORM)
C
C     INITIALIZATION
C
      ASSIGN 3100 TO NEXT
      KREF=0
      BXFAC=0.0
      IF(IBX.EQ.0) GO TO 200
      BXFAC=1.0
      DO 150 I=1,N
  150 X(I)=B(I)
  200 IF(IOP.GT.0) GO TO 1800
      IF(IBX.EQ.0) GO TO 1100
C
C     RHS MODIFICATION
C
      DO 1000 I=1,N
      II=LD(I+1)
      IF(II) 300,1000,1000
  300 BI=B(I)
      IF(BI.EQ.0.0) GO TO 1000
      II=-II
      K=I-II+IABS(LD(I))+1
      DO 900 J=K,N
      JJ=LD(J+1)
      IF(JJ) 900,900,400
  400 M=J-1
      IF(M) 500,600,600
  500 X(J)=X(J)-A(IJ+M)*BI
      GO TO 900
  600 IJ=JJ-M
      IF(IJ-IABS(LD(J))) 900,900,800
  800 X(J)=X(J)-A(IJ)*BI
  900 CONTINUE
 1000 CONTINUE
C
```

```
C      FORWARD SUBSTITUTION PASS
C
 1100 DO 1500 I=1,N
      II=LD(I+1)
      IF(II) 1200,1200,1300
 1200 X(I)=0.0
      GO TO 1500
 1300 IMI=IABS(LD(I))
      M=II-IMI-1
      X(I)=X(I)-DOTPRD(A(IMI+1),X(I-M),M)
 1500 CONTINUE
      IF(IOP.NE.0) GO TO 5000
C
C      SCALING PASS
C
 1800 DO 2000 I=1,N
      II=IABS(LD(I+1))
 2000 X(I)=A(II)*X(I)
C
C      BACKSUBSTITUTION PASS
C
      I=N
      DO 3000 K=1,N
      II=LD(I+1)
      IF(II) 2200,2200,2400
 2200 X(I)=BXFAC*B(I)
      GO TO 2800
 2400 M=II-IABS(LD(I))-1
      IF(M.EQ.0) GO TO 2800
      DO 2500 J=1,M
      X(I-J)=X(I-J)-A(II-J)*X(I)
 2500 CONTINUE
 2800 I=I-1
 3000 CONTINUE
      IF(IBX*IAFILE.EQ.0) GO TO 4000
      GO TO NEXT, (3100,3500)
C
C      ITERATIVE REFINEMENT SECTION
C
 3100 KREF=KREF+1
      IF(KREF-IREF) 3200,3200,4000
C
C      CALCULATE RESIDUAL VECTOR {R}0={B}-[A] {X} USING SKYMUL,
C      {R} RETURNS IN X, AND OLD {X} IN V
C
 3200 REWIND IAFILE
      NWA=IABS(LD(N+1))
      READ(IAFILE) (A(J),J=1,NWA)
      CALL SKYMUL (A,N,LD,X,V,-1,B,V)
C
C      SOLVE FOR CORRECTION {DX} (WHICH APPEARS IN X), CORRECT
C           OLD SOLUTION AND EVALUATE 2-NORM RELATIVE ERROR DELTA
C
      READ(IAFILE) (A(J),J=1,NWA)
      BFAC=0.0
      ASSIGN 3500 TO NEXT
      GO TO 1100
 3500 XNORM=0.0
      RNORM=0.0
      DO 3800 I=1,N
      RNORM=RNORM+X(I)*X(I)
```

```
      X(I)=V(I)+X(I)
      XNORM=XNORM+X(I)*X(I)
 3800 CONTINUE
      DELTA=0.0
      IF(XNORM.NE.0.0) DELTA=SQRT(RNORM/XNORM)
      IF(KREF-4) 3100,3100,4000
C
C     CONSTRAINED RHS RECOVERY
C
 4000 IF(IBX.LE.0) GO TO 5000
      DO 4800 I=1,N
      II=LD(I+1)
      IF(II) 4200,4200,4800
 4200 IMI=IABS(LD(I))
      M=-II-IMI-1
      B(I)=DOTPRD(A(IMI+1),X(I-M),M)
      DO 4600 J=1,N
      IJ=IABS(LD(J+1))+J-1
      IF(IJ-ABS(LD(J))) 4600,4600,4400
 4400 B(I)=B(I)+A(IJ)*X(J)
 4600 CONTINUE
 4800 CONTINUE
 5000 RETURN
      END
      SUBROUTINE SKYMUL (A,N,LD,X,AX,IOP,B,R)
      DOUBLE PRECISION AX(1),AIJ,AXI
      INTEGER LD(1)
      REAL A(1),B(1),X(1),R(1)
      DO 2000 I=1,N
      II=IABS(LD(I+1))
      AX(I)=DBLE(A(II))*DBLE(X(I))
      M=II-IABS(LD(I))-1
      IF(M.EQ.0) GO TO 2000
      DO 1500 K=1,M
      J=I-K
      AIJ=A(II-K)
      AX(I)=AX(I)+AIJ*DBLE(X(J))
      AX(J)=AX(J)+AIJ*DBLE(X(I))
 1500 CONTINUE
 2000 CONTINUE
      IF(IOP) 2500,5000,3500
 2500 DO 2800 I=1,N
      AXI=AX(I)
      R(I)=X(I)
      X(I)=DBLE(B(I))-AXI
      IF(LD(I+1).LE.0) X(I)=0.
 2800 CONTINUE
      GO TO 5000
 3500 DO 3800 I=1,N
      R(I)=DBLE(B(I))-AX(I)
      IF(LD(I+1).LE.0) R(I)=0.
 3800 CONTINUE
 5000 RETURN
      END
      SUBROUTINE SKYRDM (V,N,VMEAN,IRDM)
      REAL V(N)
      C=VMEAN-0.500
      IF(IRDM.NE.0) GO TO 300
      IX=26903
      IRDM=1
  300 DO 500 I=1,N
      IX=MOD(27181*IX+13489,65536)
```

```
      X=IX
      V(I)=.15258789E-4*X+C
  500 CONTINUE
      RETURN
      END
```

APPENDIX C.2

GASP COLUMN EQUATION SOLVER

```
      PROGRAM MAIN1 ( INPUT,OUTPUT,TAPE1,TAPE2,TAPE3,TAPE4
     *                              ,TAPE5,TAPE6,TAPE7,TAPE8 )

      COMMON  NDOF(1441)
      COMMON / A /  IQ(4),NODES,JQ(4),S(8,8)
      COMMON / SUPORT / IBD(4)


      DO 10 I=1,8
   10 CALL IOBIN ( 6HREWIND,I )

            READ  100,NDFRE,NODES,NDIVZ,NDIVY,NDIVX,IO,NOSP
      NOEL=NDIVY*NDIVX
      NOPT=(NDIVY+1)*(NDIVX+1)
      PRINT 200,NOPT,NDFRE,NOEL,NODES,NDIVZ,NDIVY,NDIVX,IO,NOSP

      CALL IOBIN(5HWRITE,6,NOPT,1)
      CALL IOBIN(5HWRITE,6,NOEL,1)
      CALL IOBIN(5HWRITE,6,NOSP,1)

      DO 1 I=1,NOPT
    1       NDOF(I)=NDFRE

      CALL IOBIN(5HWRITE,6,NDOF,NOPT)

      READ 100,JQ
      PRINT 300,JQ
      NP1=NDIVY+1
      DO 3 IX=1,NDIVX
            JQ(1)=(IX-1)*NP1+1
            JQ(4)=JQ(1)+1
            JQ(2)=JQ(1)+NP1
            JQ(3)=JQ(4)+NP1
      DO 3 IY=1,NDIVY
      DO 2 I=1,4
    2 IQ(I)=JQ(I)+IY-1
      CALL IOBIN(5HWRITE,5,IQ,4)
    3 CONTINUE

            M = NODES*NDFRE

      DO 5 I=1,M
      DO 5 J=1,M
    5       S(I,J)=1.0E-20

      DO 6 I=1,M
    6       S(I,I)=1.0

      DO 7 LZ=1,NOEL
    7       CALL IOBIN ( 5HWRITE,6,S,64 )
      DO 8 I=1,NOSP
            READ 100, IBD
    8       CALL IOBIN ( 5HWRITE,6,IBD,4 )

            CALL SECOND ( T )

      CALL JOBINFO(8,J1)
            CALL GASP1 ( IO )
      CALL JOBINFO(8,J2)

      CP=FLOAT(J2-J1)/1000.
```

PAGE 140 INTENTIONALLY BLANK

```
      CALL TIMEX ( 0,0,T )
    PRINT 400,T,CP
400 FORMAT( 6X *TIME FOR GASP  * F6.2/7X * CP FOR GASP * F6.2)

100 FORMAT ( 10I3 )
200 FORMAT ( *1* 5X *NOPT * I5/
   .             6X *NDFRE* I5/
   .             6X *NOEL * I5/
   .             6X *NODES* I5/
   .             6X *NDIVZ* I5/
   .             6X *NDIVY* I5/
   .             6X *NDIVX* I5/
   .             6X *IO   * I5/
   .             6X *NOSP * I5  )

300 FORMAT ( /// 6X *JQ   * 8I5 )


    END
    SUBROUTINE GET ( Q1,Q2,Q3,Q4,LQ,LD,L1,NUM )
    DIMENSION LQ(5000),LD(5000)
    COMMON / G /  IGET(26)
    INTEGER Q1,Q2,Q3,Q4

    CALL IOBIN ( 4HREAD,4,IGET,26 )

    Q1=IGET(1)
    Q2=IGET(2)
    Q3=IGET(3)
    Q4=IGET(4)

    L1 = NUM+1
    NUM=NUM + Q3

    I1=4
    I2=I1+Q3

    DO 1 L=L1,NUM

    I1=I1+1
    I2=I2+1

    LQ(L)=IGET(I1)
  1 LD(L)=IGET(I2)

    RETURN
    END
    SUBROUTINE TIMEX ( II,NUM,T1 )

    IF ( NUM.EQ.0 )  NUM = 1
    CALL SECOND ( T2 )

    T1 = T2 - T1

    IF ( II.EQ.0 )  GOTO 10

    FAC = NUM
    FAC = ( T1 / FAC ) * 1.0E6

    PRINT 100,II,NUM,T1,FAC

100 FORMAT ( 6X *II   * I5  ,
```

```
     •          2X *NUM * I5   ,
     •          2X *TIME* F9.2,
     •          2X *FAC * F15.2 )

  10 RETURN
     END
     SUBROUTINE GASP1 ( IO )

     COMMON IB(1441),JB(1441),NDOF(1441),IBPT(1441),R(4323)

     COMMON / A / IQ(4),NODES,JQ(4),S(8,8)

          IF ( IO.GT.0 )  PRINT 200

          CALL WIND ( 5 )
          CALL WIND ( 6 )

          CALL IOBIN ( 4HREAD,6,NOPT,1 )
          CALL IOBIN ( 4HREAD,6,NOEL,1 )
          CALL IOBIN ( 4HREAD,6,NOSP,1 )
          IF ( IO.GT.0 )  PRINT 210,NOPT,NOEL,NOSP

          CALL IOBIN ( 4HREAD,6,NDOF,NOPT )
          IF ( IO.GT.0 )  PRINT 200
          IF ( IO.GT.0 )  PRINT 220,(NDOF(I),I=1,NOPT)

          NOEQ = 0

     DO 2 I=1,NOPT

          IB   ( I ) = 0
          IBPT ( I ) = 0

  2       NOEQ = NOEQ + NDOF ( I )

          IF ( IO.GT.0 )  PRINT 230,NOEQ
          IF ( IO.GT.0 )  PRINT 200

     DO 4 LZ=1,NOEL

          CALL IOBIN ( 4HREAD,5,IQ,4 )
          IF ( IO.GT.0 )  PRINT 240,LZ,NODES,(IQ(I),I=1,NODES)

     DO 3 JJ=1,NODES

          J = IQ ( JJ )
          IB ( J ) = IB ( J ) + 1

     DO 3 II=1,NODES

          I = IQ ( II )
          IF ( I.LT.J )  GOTO 3
          L = ( I-J ) + 1
          IF ( L.GT.IBPT(J) )  IBPT ( J ) = L

  3       CONTINUE
  4       CONTINUE

          J1 = 1

     DO 6 I=1,NOPT
```

```
      J2 = J1 + NDOF ( ( ) -1
      X  = IB ( I )

  DO 5 J=J1,J2
5     R ( J ) = X

6     J1 = J2 + 1


      CALL IOBIN ( 5HWRITE,2,R,NOEQ )

      IF ( IO.GT.0 )  PRINT 200
      IF ( IO.GT.0 )  PRINT 250,(IB  (I),I=1,NOPT)
      IF ( IO.GT.0 )  PRINT 200
      IF ( IO.GT.0 )  PRINT 260,(IBPT(I),I=1,NOPT)
      IF ( IO.GT.0 )  PRINT 200

 CALL WIND ( 5 )
 CALL WIND ( 2 )

      CALL DECOM ( NOPT,IT,IO )

      IF ( IO.GT.0 )  PRINT 270,IT


      CALL SECOND ( T1 )
 CALL JOBINFO(8,J1)
 CALL SEARCH ( NOEL,NOPT,11,IO )
 CALL JOBINFO(8,J2)
 SCPH=FLOAT(J2-J1)/1000.
      CALL TIMEX  ( 0,0,T1 )

      CALL SECOND ( T2 )
 CALL JOBINFO(8,J1)
 CALL FORMQD ( NOEL,NOPT,   IO )
 CALL JOBINFO(8,J2)
 FCPD=FLOAT(J2-J1)/1000.
      CALL TIMEX  ( 0,0,T2 )

      CALL SECOND ( T3 )
 CALL JOBINFO(8,J1)
 CALL FORMDS (   NOSP,   IT,IO )
 CALL JOBINFO(8,J2)
 FCPS=FLOAT(J2-J1)/1000.
      CALL TIMEX  ( 0,0,T3 )

      CALL SECOND ( T4 )
 CALL JOBINFO(8,J1)
 CALL REDUCE ( NOPT,NOEQ,IT,IO )
 CALL JOBINFO(8,J2)
 RCPE=FLOAT(J2-J1)/1000.
      CALL TIMEX  ( 0,0,T4 )

      CALL SECOND ( T5 )
 CALL JOBINFO(8,J1)
 CALL SOLVE  ( NOPT,NOEQ,   IO )
 CALL JOBINFO(8,J2)
 SCPE=FLOAT(J2-J1)/1000.
      CALL TIMEX  ( 0,0,T5 )


     PRINT 300,T1,SCPH
```

```
      PRINT 400,T2,FCPD
      PRINT 500,T3,FCPS
      PRINT 600,T4,RCPE
      PRINT 800,T5,SCPE


300 FORMAT ( 6X *TIME FOR SEARCH* F6.2 /
     .       6X * CP  FOR SEARCH* F6.2)
400 FORMAT ( 6X *TIME FOR FORMQD* F6.2 /
     .       6X * CP  FOR FORMQD* F6.2 )
500 FORMAT ( 6X *TIME FOR FORMDS* F6.2 /
     .       6X * CP  FOR FORMDS* F6.2 )
600 FORMAT ( 6X *TIME FOR REDUCE* F6.2 /
     .       6X * CP  FOR REDUCE* F6.2 )
800 FORMAT ( 6X *TIME FOR SOLVE * F6.2 /
     .       6X * CP FOR SOLVE * F6.2 )


200 FORMAT ( *1* 10X *......GASP1......* )
210 FORMAT ( 6X *NOPT*   I5/
     .       6X *NOEL*   I5/
     .       6X *NOSP*   I5  )
220 FORMAT ( 6X *NDOF* 30I3 )
230 FORMAT ( 6X *NOEQ*   I5 )
240 FORMAT ( 6X *LZ* I5,5X *NODES* I5,5X *IQ* 20I3,5X )
250 FORMAT ( 6X *IB  * 30I3 )
260 FORMAT ( 6X *IBPT* 30I3 )
270 FORMAT ( 6X *IT*     I5 )


    RETURN
    END
    SUBROUTINE DECOM ( NOPT, IT,IO )
    COMMON IB(1441),JB(1441),NDOF(1441),IBPT(1441)
    COMMON / R / I1,I2,K1,K2,MO
    DIMENSION IC(5)
    EQUIVALENCE ( I1,IC )
            IP = 1
            IT = 1
            I1 = 1
            I2 = IBPT ( 1 )
  1 L = IP + IBPT ( IP ) - 1
            IF ( L.LT.NOPT )  CALL UPDOF ( L,IP,K1,K2,MO )

         CALL IOBIN ( 5HWRITE,3,IC,5 )


         IF ( IO.GT.0 )  PRINT 10,I1,I2,K1,K2,MO
            IF ( L.GE.NOPT )  GOTO 4

         ICHEK = IBPT ( IP )

  2 IU = IBPT(IP+1)-ICHEK
            IF ( IU.GE.0 )  GOTO 3

         ICHEK = ICHEK - 1

            IP = IP + 1
            I2 = I2 - 1
    GOTO 2
  3         I1 = I2
            I2 = I2 + IU
```

```
            IP = IP + 1
            IT = IT + 1
      GOTO 1


   10  FORMAT (     6X *I1* I6,
     .              2X *I2* I6,
     .              2X *K1* I6,
     .              2X *K2* I6,
     .              2X *MO* I6    )
    4 RETURN
      END
      SUBROUTINE UPDOF ( L,IP,K1,K2,MO )
      COMMON IB(1441),JB(1441),NDOF(1441),IBPT(1441)

           ICHEK = IBPT ( IP )

              ID = IP
              K1 = NDOF ( ID ) + 1
              K2 = 0
              MO = 1
      DO   1  J=ID,L
    1         K2 = K2 + NDOF ( J )

    2 IU = IBPT ( ID+1 ) - ICHEK
              IF ( IU.GE.0 )  GOTO 3

           ICHEK = ICHEK - 1

              ID = ID + 1
              K1 = K1 + NDOF ( ID )
              MO = MO + 1

      GOTO 2

    3 RETURN
      END
      SUBROUTINE SEARCH (NOEL,NOPT,IBANDP,IO )
      COMMON IB(1441),JB(1441),NDOF(1441),LQ(11,51),LD(11)
      COMMON /A/ IQ(4),NODES,JQ(4),S(8,8)
      DO 1 J=1,NOPT
    1      JB(J) = IB(J)
           IF ( IO.GT.1 ) PRINT 200
  200      FORMAT (1H1 5X *SEARCH*)
      DO 13 I=1,IBANDP
            LD ( I ) = 0
      DO 13 J=1,51
   13       LQ(I,J)=0
            IK = 1

      DO 11 LZ=1,NOEL
           CALL IOBIN ( 4HREAD,5,IQ,4 )
      DO 3 II=1,NODES
           I =IQ(II)
           IB(I) = IB(I) - 1
      DO 3 JJ=1,NODES
           J = IQ(JJ) - IK + 1
           IF ( IQ(JJ).GT.I) GOTO 3
      DO 2 LL=1,IBANDP
           IF ( LQ(LL,J).EQ.0 ) LQ(LL,J)=I
           IF ( LQ(LL,J).EQ.I ) GOTO 3
    2 CONTINUE
```

```
    3 CONTINUE
            IF ( IB(IK).EQ.0 )  CALL ORDER ( IK,IBANDP,IO,NOPT )
      PRINT 379,LZ
  379 FORMAT(5X,*LZ*,I6)
   11 CONTINUE

      CALL WIND ( 4 )
      CALL WIND ( 5 )

      RETURN
      END
      SUBROUTINE ORDER ( IK,IBANDP,IO,NOPT )
      COMMON IB(1441),JB(1441),NDOF(1441),LQ(11,51),LD(11)
      COMMON / G / IGET(26)

    4 DO 5 I=1,IBANDP
            IF ( LQ(I,1).EQ.0 ) GOTO 6
    5 IT = I
    6 DO 8 I=1,IT
            MIN = LQ(I,1)
            LOC = I
      DO 7 J=I,IT
            IF ( LQ(J,1).GT.MIN) GOTO 7
            MIN = LQ(J,1)
            LOC = J
    7 CONTINUE
            LQ(LOC,1) = LQ(I,1)
            LQ(  I,1) = MIN
    8 CONTINUE
            LD(1) = 1
            IF ( IT.EQ.1 ) GOTO 14
      DO 9 L=2,IT
            I = LQ(L-1,1)
    9       LD(L) = LD(L-1) + NDOF(I)*NDOF(IK)
   14       I = LQ(IT,1)
            MD= LD(IT) + NDOF(I)*NDOF(IK) - 1
            IF ( IO.GT.1 ) PRINT 1300,IT,MD
            IF ( IO.GT.1 ) PRINT 1200,(LQ(I,1),I=1,IT)
            IF ( IO.GT.1 ) PRINT 1250,(LD(I),I=1,IT)

            IGET(1)=JB(IK)
            IGET(2)=NDOF(IK)
            IGET(3)=IT
            IGET(4)=MD

            I1=4
            I2=I1+IT

      DO 90 I=1,IT
            IGET(I+I1)=LQ(I,1)
   90       IGET(I+I2)=LD(I  )
            CALL IOBIN ( 5HWRITE,4,IGET,26 )

            IF ( IK.EQ.NOPT ) GOTO 12
      DO 10 J=1,50
      DO 10 I=1,IBANDP
   10       LQ(I,J) = LQ(I,J+1)
            IK = IK + 1
            IF ( IB(IK).EQ.0 ) GOTO 4

 1300 FORMAT ( 5X *MQ* I5 /
     .         5X *MD* I5   )
```

```
1200      FORMAT (5X *LQ* 16I5)
1250      FORMAT (5X *LD* 16I5)
  12 RETURN
     END
     SUBROUTINE FORMQD (NOEL,NOPT,IO )
     COMMON I8 (335),NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),
    . D(37000)
     COMMON /A/ IQ(4),NODES,JQ(4),S(8,8)
     INTEGER DUM
     IF ( IO.GT.0 ) PRINT 200
 200 FORMAT (1H1 5X *FORMQD*)
          IK = 0
          NUM = 0
          DUM = 0
          JOUT = 1
     DO 4 LZ=1,NOEL

          CALL IOBIN ( 4HREAD,5,IQ,4    )
          CALL IOBIN ( 4HREAD,6,S     ,64 )
          J = IQ(1) - IK
     DO 6 JJ=1,NODES
          K = IQ ( JJ ) - IK
          IF ( K.GT.J )  J = K
   6      CONTINUE
          IF (J.GE.JOUT) CALL EXPAND (IK,J,JOUT,NUM,DUM)

     IF ( IO.GT.0 ) PRINT 1149,LZ,NUM,DUM

     DO 7 JJ=1,NODES
          K = IQ ( JJ ) - IK
   7      JQ ( JJ ) = NDOF ( K )



     DO 1 II=1,NODES
          I = IQ(II) - IK
          I8(I) = I8(I) - 1
     DO 1 JJ=1,NODES
          J = IQ(JJ) - IK
          IF (J.GT.I) GOTO 1
          CALL FIND1(I,J,LOQ,LOD)
          CALL ADDS (II,JJ,LOD)
   1 CONTINUE


   2 IF ( I8(1).GT.0  .OR.  IK.EQ.NOPT )  GOTO 4
          LT = MD ( 1 )
          CALL IOBIN ( 5HWRITE,7,D,LT )
          CALL SHRINK ( IK,1,JOUT,NUM,DUM )




     GOTO 2


   4 CONTINUE

     CALL WIND ( 3 )
     CALL WIND ( 4 )
     CALL WIND ( 7 )
```

```
1149 FORMAT ( 5X *LZ    *  I5,
     .        5X *NUM   *  I5,
     .        5X *DUM   *  I5 )
      RETURN
      END
      SUBROUTINE EXPAND ( IK, J,JOUT,NUM,DUM )
      COMMON IB(335),NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),
     . D(37000)
      INTEGER DUM
      DO 3 N=JOUT,J

           CALL GET ( IB(N),NDOF(N),MQ(N),MD(N),LQ,LD,L1,NUM )

      DO 1 L=L1,NUM
    1      LQ(L) = LQ(L) - IK
           L1 = DUM + 1
           DUM = DUM + MD(N)
      DO 2 L=L1,DUM
    2      D(L) =0.0
    3      CONTINUE
           JOUT = J + 1
           RETURN
           END

      SUBROUTINE FIND1( I,J,LOQ,LOD )
      COMMON IB(335),NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),
     . D(37000)

           LOQ = 0
           LOD = 0
           L   = 0
      IF (J.EQ.1) GOTO 2
           M2 = J-1
      DO 1 M=1,M2
           L   = L   + MQ(M)
    1      LOD = LOD + MD(M)
    2      L1 = L + 1
           L2 = L + MQ(J)
      DO 3 LL=L1,L2
           IF ( LQ(LL).EQ.I ) LOQ = LL
    3      CONTINUE
           LOD = LD(LOQ) + LOD
           IF ( LOQ.EQ.0 ) PRINT 90,I,J,LOQ,LOD
   90      FORMAT ( 5X *ERROR IN FIND* / 5X,4I5/5X,2I5)
           RETURN
           END

      SUBROUTINE ADDS ( II,JJ,LOD )
      COMMON IB(335),NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),
     . D(37000)
      COMMON /A/ IQ(4),NODES,JQ(4),S(8,8)
      M1 = 1
      IF (II.EQ.1) GOTO 2
      I2 = II - 1
      DO 1 I=1,I2
    1 M1 = M1 + JQ(I)
    2 M2 = M1 + JQ(II) - 1
      N1 = 1
      IF (JJ.EQ.1) GOTO 4
      J2 = JJ - 1
      DO 3 J=1,J2
    3 N1 = N1 + JQ(J)
```

```
4 N2 = N1 + JQ(JJ) - 1
  DO 5 M=M1,M2
  DO 5 N=N1,N2
       D(LOD) = D(LOD) + S(M,N)
5      LOD    = LOD + 1
  RETURN
  END

  SUBROUTINE SHRINK ( IK,MO,JOUT,NUM,DUM )
  COMMON IB(335),NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),
 . D(37000)
  INTEGER DUM



       IT = MQ ( 1 ) + 1
       LT = MD ( 1 ) + 1
       K = 0
       J1 = MO + 1
       J2 = JOUT - 1

  DO 2 J=J1,J2
       K = K + 1
       IB(K) = IB(J)
       NDOF(K) = NDOF(J)
       MQ(K) = MQ(J)
2      MD(K) = MD(J)
       K = 0
  DO 3 J= IT,NUM
       K = K+ 1
       LQ(K) = LQ(J) - MO
3      LD(K) = LD(J)
       K = 0
  DO 4 J=LT,DUM
       K = K + 1
4      D(K) = D(J)
       JOUT = JOUT - MO
       NUM = NUM - IT +1
       DUM = DUM - LT + 1
       IK  = IK  + MO
  RETURN
  END
  SUBROUTINE FORMDS ( NOSP,IT,IO)
  COMMON NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),D(37000),
 . DS(100)
  COMMON /R/ I1,I2,K1,K2,MO
  COMMON /SUPORT/ IBD(4)
  DIMENSION IC(5)
  EQUIVALENCE ( I1,IC )
  INTEGER DUM,ROW,RUN

       IF ( IO.GT.0 )  PRINT 2000

       IK = 0
       NUM = 0
       DUM = 0
       ROW = 0
       CALL IOBIN ( 4HREAD,6,IBD,4 )
       RUN = 1
 DO 1 IR=1,IT

      CALL IOBIN ( 4HREAD ,3,IC,5 )
```

```
            CALL IOBIN ( SHWRITE,1,IC,5 )



            CALL EXPLOD (IK,NUM,DUM)

            IF ( IO.GT.0 )  PRINT 3000,IR,NUM,DUM

            CALL STORE (NOSP,RUN,ROW)
      IF ( IR.EQ.IT ) GOTO 1
            CALL COMPACT (NUM,DUM)
    1 CONTINUE
      CALL WIND ( 1 )

2000 FORMAT ( *1     FORMDS * )
3000 FORMAT ( 5X *IR* I6,2X *NUM* I6,2X *DUM* I6 )


      RETURN
      END
      SUBROUTINE EXPLOD ( IK,NUM,DUM )
      COMMON NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),D(37000),
     . DS(100)
      COMMON /R/ I1,I2,K1,K2 ,MO
      INTEGER DUM
      DO 3 N=I1,I2

            CALL GET ( IB,NDOF(N),MQ(N),MD(N),LQ,LD,L1,NUM )

      DO 1 L=L1,NUM
    1       LQ(L) = LQ(L) - IK
            L2 = DUM + 1
            DUM = DUM + MD(N)

            LIM=DUM-L2+1
            CALL IOBIN ( 4HREAD,7,D(L2),LIM )

    3       CONTINUE
            IK = IK + MO
            RETURN
            END

      SUBROUTINE STORE (NOSP,RUN,ROW)
      COMMON NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),D(37000),
     . DS(100)
      COMMON / SUPORT / IBD(4)
      COMMON /R/ I1,I2,K1,K2,MO
      INTEGER ROW,RUN
      DO 9 I=I1,I2
            MBAND = 0
      DO 1 J=1,I
    1       MBAND = MBAND + NDOF(J)
      DO 2 J=1,I
            J1 = J
            CALL FIND2(I,J,LOQ,LOD)
            IF ( LOQ.GT.0 )  GOTO 3
    2       MBAND = MBAND - NDOF(J)
    3       NWORDS = MBAND*NDOF(I)
      DO 4 K=1,NWORDS
    4       DS(K) = 0.0
```

```
            M1 = 1
      DO 6 J= J1,I
            CALL FIND2( I,J,LOQ,LOD )
            IF ( LOQ.EQ.0 )  GOTO 6
            M2 = M1 + NWORDS - MBAND
            NJ = NDOF(J) - 1
      DO 5 M=M1,M2,MBAND
            N2 = M + NJ
      DO 5 N=M,N2
            DS ( N ) = DS ( N ) + D ( LOD )
5           LOD      = LOD + 1
6           M1 = M1 + NDOF ( J )
            ROW = ROW + 1
      IF ( ROW.NE.IBD(1) ) GOTO 8

            M1 = MBAND  - NDOF(I)
            M2 = NWORDS - NDOF(I)
            LOC = 1

      DO 7 M = M1,M2,MBAND
            IF ( IBD(LOC+1).GT.0 ) DS(M+LOC) = 1.E50
7           LOC = LOC + 1

            IF ( RUN.LT.NOSP ) CALL IOBIN ( 4HREAD,6,IBD,4 )
            RUN = RUN + 1


8           CALL IOBIN ( 5HWRITE,1,NDOF(I),1 )
            CALL IOBIN ( 5HWRITE,1,NWORDS ,1 )
            CALL IOBIN ( 5HWRITE,1,DS     ,NWORDS )

9     CONTINUE
      RETURN
      END

      SUBROUTINE FIND2( I,J,LOQ,LOD )
      COMMON NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),D(37000),
     . DS(100)

            LOQ = 0
            LOD = 0
            L   = 0
      IF (J.EQ.1) GOTO 2
            M2 = J-1
      DO 1 M=1,M2
            L   = L   + MQ(M)
1           LOD = LOD + MD(M)
2           L1 = L + 1
            L2 = L + MQ(J)
      DO 3 LL=L1,L2
            IF ( LQ(LL).EQ.I ) LOQ = LL
3           CONTINUE
            LOD = LD(LOQ) + LOD
            RETURN
            END
      SUBROUTINE COMPACT ( NUM,DUM )
      COMMON NDOF(335),MQ(335),MD(335),LQ(5000),LD(5000),D(37000),
     . DS(100)
      COMMON /R/ I1,I2,K1,K2,MO
      INTEGER DUM
      IT = 1
      LT = 1
```

```
      DO 1 J=1,MO
           IT= IT + MQ(J)
    1      LT= LT + MD(J)
           K = 0
           J1 = MO + 1
           J2 = I2

      DO 2 J=J1,J2
           K = K + 1
           NDOF(K) = NDOF(J)
           MQ(K) = MQ(J)
    2      MD(K) = MD(J)
           K = 0
      DO 3 J= IT,NUM
           K = K+ 1
           LQ(K) = LQ(J) - MO
    3      LD(K) = LD(J)
           K = 0
      DO 4 J=LT,DUM
           K = K + 1
    4      D(K) = D(J)
           NUM = NUM - IT +1
           DUM = DUM - LT + 1
      RETURN
      END
      SUBROUTINE REDUCE ( NOPT,NOEQ,IT,IO )
      COMMON D(33000),JD(300),KD(300),MB(300),NUM
      COMMON /R / I1,I2,K1,K2,MO
      COMMON / TRANS /  DS(100),NDOF,NOEL,IB,LCPJ(1441)
      DIMENSION IC(5)
      EQUIVALENCE ( I1,IC )

      LOC = 0

           M1 = 1
           L1 = 1

      DO 4 IR=1,IT

           NCAL=0

           CALL SECOND ( Q1 )

           CALL IOBIN ( 4HREAD,1,IC,5 )

C          IF ( IO.GT.0 ) PRINT 200,IR,I1,I2

      DO 3 II=I1,I2
           CALL INITAL (M1,L1,MD,M2,LD,L2,NDOF,NOEL,IB)
           L   = IB - NDOF - 2
           IS  = KD(MD)
           ILS = JD(MD)
           ILT = JD(M2)
           IE  = -JD(IS+1) - IS + KD(IS+1)

      IF ( IB + NDOF .NE. 3 )  GOTO 10
           D ( ILS  ) = D ( ILS ) / D ( ILS-1 )
           D ( ILS+1) = SQRT ( D ( ILS + 1 ) - D ( ILS )**2 )

           NUM = 1

      GOTO 2
```

```
10        NUM = 0
      IF ( IB.GT.NDOF ) GOTO 1

          NUM = 0

          D(ILS) = SQRT ( D(ILS) )
      IF (NDOF.EQ. 1)   GOTO 2

          I = ILS + NDOF
          D(I) = D(I) / D(ILS)
          D(I+1) = SQRT ( D(I+1) - D(I)**2 )
          ILS = ILS + 2*NDOF
          L = L + 2

          NUM = NUM + 1

          IF (NDOF.EQ.2) GOTO 2
 1        CALL SECOND (T1)
          CALL CHOL4 ( L,IS,ILS,ILT,IB,IE )
C         CALL TIMEX  ( II,NUM,T1 )

 2        NCAL = NCAL + NUM

          LOC = LOC + 1
          L   = 0

      DO 20 N=LD,L2
          L = L + 1
 20       DS ( L ) = D ( N )

          CALL IOBIN ( 6HWRITER,8,DS,103,LCFJ ( LOC ) )


 3  CONTINUE

      IF ( IO.GT.0 )  PRINT 300,L2

            IF ( IR.LT.IT )  CALL UPDATE ( K1,K2,M1,L1 )
          CALL TIMEX ( IR,NCAL,Q1 )

 4  CONTINUE

          CALL WIND ( 8 )

100 FORMAT ( *1* 5X *REDUCE* )
200 FORMAT ( *1* 5X *IR* I5,4X *I1* I5,4X *I2* I5,/ )
300 FORMAT ( / 5X *STORAGE FOR D* I7 / )
    RETURN
    END

    SUBROUTINE INITAL ( M1,L1,MD,M2,LD,L2,NDOF,NOEL,IB )
    COMMON D(33000),JD(300),KD(300),MB(300)
C   THIS SUBROUTINE ESTABLISHES: D--L1,L2,AND JD,KD,MB--M1,M2
    MD = M1
    LD = L1

          CALL IOBIN ( 4HREAD,1,NDOF,1 )
          CALL IOBIN ( 4HREAD,1,NOEL,1 )
    IB = NOEL /NDOF
```

```
      L2 = L1 + NOEL - 1

           LIM=L2-L1+1
           CALL IOBIN ( 4HREAD,1,D(L1),LIM )

      M2 = M1 + NDOF - 1
      L1 = L1 - IB
      DO 1 M=M1,M2
           L1 = L1 + IB
           JD(M) = L1
           KD(M) = M2 - IB
    1      MB(M) = IB
      M1 = M2 + 1
      L1 = L1 + IB
      RETURN
      END
      SUBROUTINE CHOL4 ( L,IS,ILS,ILT,MBAND,IE )
      COMMON D(33000),JD(300),KD(300),MB(300),NUM


      DO 4 I=ILS,ILT,MBAND

           L  = L + 1
           M  = IS+ 1
           ID = I + IE
           J1 = I + 1
           J2 = I + L

           D(I) = D(I) / D(I-ID)

      DO 2 J=J1,J2

           M  = M + 1
           IC = IS- KD(M)
           ID = I - JD(M)-IC
                                K1 = I
           IF ( IC.LT.0 )       K1 = K1-IC
                                K2 = J -1

           SUM = 0.0

      DO 1 K=K1,K2
C          NUM = NUM + 1
    1      SUM = SUM + D(K) * D(K-ID)

    2      D(J) = ( D(J)-SUM ) / D(J-ID)

           SUM = 0.0

      DO 3 K=I,J2
C          NUM = NUM + 1
    3      SUM = SUM + D(k)*D(K)

    4      D(J2+1) = SQRT ( D(J2+1)-SUM )

      RETURN
      END
      SUBROUTINE UPDATE ( I1,I2,M1,L1 )
      COMMON D(33000),JD(300),KD(300),MB(300)
           I3 = I1 - 1

           NUM = 0
```

```
      DO 2 I=I1,I2

           IC = KD(I) - I3
                              J1 = JD(I)
           IF ( IC.LT.0 )     J1 = J1 - IC
                              J2 = JD(I) + MB(I) - 1

           DO 1 J=J1,J2
                NUM = NUM + 1
    1           D(NUM) = D(J)

           L = I - I3
           JD(L) = NUM + J1 - J2
           KD(L) = KD(I) - I3
           IF ( IC.LT.0 ) KD(L) = 0
                              MB(L) = MB(I)
           IF ( IC.LT.0 )     MB(L) = MB(L) - I3


    2 CONTINUE


      M1 = M1 - I3
      L1 = NUM + 1


      RETURN
      END
      SUBROUTINE SOLVE ( NOPT,NOEQ,IO )
      COMMON R(4323)

           CALL IOBIN ( 4HREAD,2,R,NOEQ )

      IF ( IO.GT.0 )  PRINT 100
      IF ( IO.GT.0 )  PRINT 200,(R(I),I=1,NOEQ)
           CALL SECOND ( T1 )
      CALL FPASS ( NOPT,JE )


           CALL TIMEX ( 0,0,T1 )
           CALL SECOND ( T2 )
           CALL BPASS ( NOPT,JE )
           CALL TIMEX  ( 0,0,T2 )

      IF ( IO.GT.0 )  PRINT  100
      IF ( IO.GT.0 )  PRINT  200,(R(I),I=1,NOEQ)

      PRINT 300,T1
      PRINT 400,T2

  100 FORMAT ( *1* 5X *SOLVE* )
  200 FORMAT ( 5X *R  * 20F4.1 )
  300 FORMAT ( *1* 5X *TIME FOR FPASS* F6.2 )
  400 FORMAT (     6X *TIME FOR BPASS* F6.2 )

      RETURN
      END
      SUBROUTINE FPASS (NOPT,JE)
      COMMON R(4323)
      COMMON / TRANS /  D(100),NDFRE,NWORDS,MBAND,LCPJ(1441)
      JE = 0
```

```
      DO 3 NODE=1,NOPT

           CALL IOBIN ( 7HREADSKP,8,D,103,LCPJ( NODE ) )

           PRINT 200,NDFRE,NWORDS,MBAND
  200 FORMAT ( 3I5 )

           JE = JE + NDFRE
           JS = JE - MBAND + 1
           I2 = (NDFRE - 1) * MBAND + 1
           M  = MBAND - NDFRE - 2


      DO 3 I=1,I2,MBAND

           M = M + 1
           J2= I + M
           ID= JS - I
           SUM = 0.0


           IF ( I.GT.J2 ) GOTO 2

      DO 1 J=I,J2
    1 SUM = SUM + D(J) * R(J+ID)


    2 J = J2 + 1

      R(J+ID) = ( R(J+ID)-SUM ) / D(J)


    3 CONTINUE

      RETURN
      END
      SUBROUTINE BPASS (NOPT,JE)
      COMMON R(4323)
      COMMON / TRANS /  D(100),NDFRE,NWORDS,MBAND,LCPJ(1441)


      DO 4 NODE=1,NOPT

           LOC = NOPT-NODE+1
           CALL IOBIN ( 7HREADSKP,8,D,103,LCPJ ( LOC ) )
      JT = JE
           JS = JT-MBAND+1


      DO 2 I=1,NDFRE

           J1 = (NDFRE - I) * MBAND + 1
           J2 = J1 + MBAND - I - 1
           R(JT) = R(JT) / D(J2+1)
           ID = JS - J1


      IF (J1.GT.J2) GOTO 3

      DO 1 J=J1,J2
```

```
1 R(J+ID) = R(J+ID) - D(J) * R(JT)

2       JT = JT - 1


3 JE = JE - NDFRE

4 CONTINUE

  RETURN
  END
  SUBROUTINE WIND ( NTAPE )
1 CALL IOBIN ( 6HWRITER,NTAPE )
2 IF ( IOBIN(4HTEST,NTAPE)) 2,3,3
3 CALL IOBIN ( 6HREWIND,NTAPE )
  RETURN
  END
```

APPENDIX C.3

COLSOL COLUMN EQUATION SOLVER

```
      PROGRAM TEST (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE1,TAPE2,
     1              TAPE3,TAPE4,TAPE7,TAPE8)
      COMMON S(20,20),R(20,5),A(50),MA(20),KA(4),B(50),MB(20),KB(4),
     1       X(20,5),MX(20),KX(4),KH(20),NB(20),NEB(10),NEBL(10),
     2       D(20),NT(6)
      MAXT=50
      MAXC=20
      DO 5 N=1,4
    5 NT(N)=N
      NT(5)=7
      NT(6)=8
   10 CONTINUE
      READ (5,1000) NEQ,LEQ,NBLK,NBLL,NLD
      IF (NEQ.EQ.0) STOP
      WRITE(6,1000) NEQ,LEQ,NBLK,NBLL,NLD
C     READ S  UPPER TRIANG   FULL
      DO 100 I=1,NEQ
      READ (5,2000) (S(J,I),J=1,I)
      DO 100 J=1,I
  100 S(I,J)=S(J,I)
      CALL PRMAT (S,NEQ,NEQ,20,1HS)
      DO 120 I=1,NLD
  120 READ (5,2000) (R(J,I),J=1,NEQ)
      CALL PRMAT (R,NEQ,NLD,20,1HR)
C     PROFILE S
      DO 200 I=1,NEQ
      KH(I)=I
      IM=I-1
      DO 180 J=1,IM
      IF (S(I,J)) 200,170,200
  170 KH(I)=I-J
  180 CONTINUE
  200 CONTINUE
      WRITE (6,1000) (KH(I),I=1,NEQ)
      READ (5,1000) (NEB(I),I=1,NBLK)
      WRITE(6,1000) (NEB(I),I=1,NBLK)
      NL=0
      DO 500 N=1,NBLK
      NF=NL+1
      NL=NF+NEB(N)-1
      KA(1)=N
      KA(3)=NF
      KA(4)=NL
      KA(2)=N
      L=0
      DO 300 I=NF,NL
      NB(I)=N
      JF=I-KH(I)+1
      KHI=NEQ-JF+1
      DO 250 I=JF,I
      L=L+1
  250 A(L)=S(J,I)
      MA(I-NF+1)=L
C     FIND LOWEST PREV. BLOCK TO OPERAPE
      IF (N.EQ.1) GO TO 500
      KA(2)=MIN0 (KA(2),NB(NEQ-KHI+1))
  300 CONTINUE
      CALL TAPES (NT(1),N,A,MAXT,MA,MAXC,KA,2)
      WRITE (6,3000) N,KA
      WRITE (6,9000) (MA(I-NF+1),I=NF,NL)
 9000 FORMAT (4H MA ,16I5)
```

```
      CALL PRAB (A,B,KA,KB,MA,MB)
  500 CONTINUE
      READ (5,1000)(NEBL(I),I=1,NBLL)
      WRITE(6,1000)(NEBL(I),I=1,NBLL)
      NL=0
      DO 600 N=1,NBLL
      NF=NL+1
      NL=NF+NEBL(N)-1
      KA(1)=N
      KA(2)=0
      KA(3)=NF
      KA(4)=NL
      L=0
      DO 550 I=NF,NL
      DO 540 J=1,NEQ
      L=L+1
  540 A(L)=R(J,I)
C 550 MA(I-NF+1)=L
  550 MA(I-NF+1)=0                                                    **********
      CALL TAPES (NT(3),N,A,MAXT,MA,MAXC,KA,2)
      WRITE (6,4000) N,KA
      CALL  PRR (A,MA,KA,NEQ)
  600 CONTINUE
      IF (LEQ.EQ.NEQ) GO TO 650
C     SET SOLN LEQ+1 TO NEQ ON TAPE NT(5)
      WRITE (6,9991)
 9991 FORMAT (30H SOLVE SUBSTRUCT DISPLACEMENTS  )
      KKK=1
      CALL SORE(A,B,D,MA,MB,NEQ,NEQ,NBLK,NBLL,MAXT,MAXC,NT,KKK)
C     SET SUBSTR DISPLS ON TAPE NT(5)
      DO 630 N=1,NBLL
      CALL TAPES (NT(4),N,A,MAXT,MA,MAXC,KA,1)
      NTA=KA(4)-KA(3)+1
      LEQQ=LEQ+1
      IC=0
      L=0
      DO 620 I=1,NTA
      DO 610 K=LEQQ,NEQ
      L=L+1
  610 B(L)=A(K+IC)
  620 IC=IC+NEQ
      CALL TAPES (NT(5),N,B,L,MB,MAXC,KA,2)
  630 CONTINUE
  650 CONTINUE
      WRITE (6,9992)
 9992 FORMAT (13H SOLVE SYSTEM   )
      KKK=1
      NTS=4
      KKK=3
      KKK=4
      KKK=5
      NTS=3
      KKK=2
      CALL SORE(A,B,D,MA,MB,NEQ,LEQ,NBLK,NBLL,MAXT,MAXC,NT,KKK)
      KKK=5
      NTS=3
      IF (LEQ.GE.NEQ) GO TO 850
      IF (KKK.NE.5) GO TO 850
C
C      SET SUBSTR DISPLACEMENTS IN REQ LOAD BLOCKS ON TAPE NT(4)
C
      DO 830 N=1,NBLL
```

```
C       READ RED LOADS
        CALL TAPES (NT(4),N,A,MAXT,MA,MAXC,KA,1)
        NTA=KA(4)-KA(3)+1
        LL=NTA*(NEQ-LEQ)
        CALL TAPES (NT(5),N,B,LL,MB,MAXC,KBM1)
        IC=0
        JC=0
        LEQQ=LEQ+1
        DO 820 I=1,NTA
        DO 810 J=LEQQ,NEQ
  810 A(IC+J)=B(JC+J-LEQ)
        IC=IC+NEQ
  820 JC=JC+NEQ-LEQ
        CALL TAPES (NT(6),N,A,MAXT,MA,MAXC,KA,2)
  830 CONTINUE
        NT(4)=NT(6)
  850 CONTINUE
        CALL SORE(A,B,D,MA,MB,NEQ,LEQ,NBLK,NBLL,MAXT,MAXC,NT,KKK)
        DO 800 N=1,NBLL
        CALL TAPES (NTS,N,A,MAXT,MA,MAXC,KA,1)
        WRITE (6,6000) N
        CALL PRR (A,MA,KA,NEQ)
C       FORM S*DISP
        NL=KA(4)-KA(3)+1
        DO 750 L=1,NL
        DO 740 I=1,NEQ
        SS=0.0
        LC=(L-1)*NEQ
        DO 730 J=1,NEQ
  730 SS=SS+S(I,J)*A(J+LC)
  740 X(I,L)=SS
  750 CONTINUE
        WRITE (6,7000) N
        CALL PRMAT (X,NEQ,NL,20,1HX)
  800 CONTINUE
        GO TO 10
 1000 FORMAT (16I5)
 2000 FORMAT (16F5.0)
 3000 FORMAT (/6H BLOCK  ,I5/4H KA  ,4I5)
 4000 FORMAT (/11H LOAD BLOCK  ,I5/4H KA  ,4I5)
 6000 FORMAT (/15H SOLUTION BLOCK  ,I5)
 7000 FORMAT (/12H K*X...BLOCK ,I5)
        END
        SUBROUTINE PRMAT (A,NR,NC,MM,H)
        DIMENSION A(MM,1)
        WRITE (6,2000) H
 2000 FORMAT (/9H MATRIX   ,A5)
        DO 200 N=1,NC,8
        NL=N+7
        IF (NL.GT.NC) NL=NC
        DO 200 I=1,NR
  200 WRITE (6,1000) I,(A(I,J),J=N,NL)
        RETURN
 1000 FORMAT (I5,8F14.6)
        END
        SUBROUTINE PRR (R,MR,KR,NEQ)
        DIMENSION R(1),MR(1),KR(4)
        NL=KR(4)-KR(3)+1
        II=0
        DO 100 N=1,NL
        WRITE (6,1000) N,(R(I+II),I=1,NEQ)
  100 II=II+NEQ
```

```
      RETURN
 1000 FORMAT (3H R ,I4,14F9.2)
      END
      SUBROUTINE SORE (A,B,D,MA,MB,NEQ,LEQ,NBLK,NBLL,MAXT,MAXC,NT,KKK)
C-----SOLOTION OR REDUCTION OF LINEAR EQUATIONS STORED OUT OF CORE IN
C     COMPACTED ACTIVE COLUNM BLOCKS OF APPROXIMATELY MAXT LOCATIONS.
C     PROGRAMMED BY E WILSON AND H DOVEY  JAN 1976
      DIMENSION A(MAXT),B(MAXT),D(NEQ),MA(MAXC),MB(MAXC),NT(4)
      DIMENSION KA(4),KB(4)
C     ARRAYS A AND B ARE WORKING STORAGE AREAS FOR BLOCKS OF COLUMNS
C     OF THE COEFFICIENT MATRIX OR LOAD VECTORS. WHERE MAXC IS THE
C     MAXIMUM NUMBER OF COLUMNS OR VECTORS IN A BLOCK.  MA AND MB ARE
C     INTEGER ARRAYS OF LOCATION OF DIAGONAL TERMS IN THE A OR B ARRAYS.
C     THE D ARRAY STORES DIAGONAL TERMS OF REDUCED MATRIX .
C     KA(1),KB(1)= BLOCK NUMBER OF A OR B BLOCK
C     KA(2),K(,2)= NUMBER OF LOWEST BLOCK TO OPERATE ON THIS BLOCK.
C     KA(3),KB(3)= NUMBER OF FIRST COLUMN IN BLOCK
C     KA(4),KB(4)= NUMBER OF LAST COLUMN IN BLOCK
C     NBLK    NUMBER OF BLOCKS OF COEFFICIENT MATRIX TERMS
C     NBLL    NUMBER OF BLOCKS OF LOAD VECTORS
C     NEQ     NUMBER OF EQUATIONS IN COEFFICIENT MATRIX
C     LEQ     NUMBER OF LAST EQUATION TO BE REDUCED IN COEFFICIENT MATRIX
C     NT(1)   TAPE NUMBER FOR STORAGE OF BLOCKS OF THE COEFFICIENT MATRIX
C     NT(3)   TAPE NUMBER FOR STORAGE OF BLOCKS OF THE LOAD VECTORS
C     NT(4)   TAPE NUMBER FOR STORAGE OF BLOCKS OF THE DISPLACEMENT OR
C             REDUCED LOAD VECTORS
C     KKK=1 COMPLETE SOLUTION - REQUIRES LEQ=NEQ
C     KKK=2 FORWARD REDUCTION OF COEFFICIENT MATRIX AND LOAD VECTORS
C     KKK=3 FORWARD REDUCTION OF COEFFICIENT MATRIX ONLY
C     KKK=4 FORWARD REDUCTION OF LOAD VECTORS ONLY
C     KKK=5 BACKSUBSTITUTION ONLY. IF LEQ IS LESS THAN NEQ SUBSTRUCTURE
C           DISPLACEMENT  MUST BE PREVIOUSLY CALCULATED
      GO TO (100,100,100,300,300) KKK
C-----BLOCK-BY-BLOCK TRIANGULARIZATION OF MATRIX
  100 DO 200 N=1,NBLK
C     1. MOVE PREVIOUSLY REDUCED BLOCK TO B
      IF (N.EQ.1) GO TO 110
      CALL BLKOP (A,B,D,MA,MB,KA,KB,NEQ,LEQ,MAXT,MAXC,1)
C     2. READ NEW BLOCK FROM TAPE NT(1)
  110  CALL TAPES (NT(1),N,A,MAXT,MA,MAXC,KA,1)
C     3. OPERATE ON BLOCK N WITH BLOCK M
      M = KA(2)
      IF (M.EQ.N) GO TO 160
      NM=N-1
      IF (M.EQ.NM) GO TO 140
C     READ BLOCK M FROM TAPE NT(2)
  120 CALL TAPES (NT(2),M,B,MAXT,MB,MAXC,KB,1)
  140 CALL BLKOP (A,B,D,MA,MB,KA,KB,NEQ,LEQ,MAXT,MAXC,2)
      M = M+1
      IF (M.LT.N) GO TO 120
C     4. SELF-REDUCTION OF BLOCK N
  160 CALL BLKOP (A,A,D,MA,MA,KA,KA,NEQ,LEQ,MAXT,MAXC,3)
C     5. WRITE REDUCED BLOCK N ON TAPE NT(2)
      CALL TAPES (NT(2),N,A,MAXT,MA,MAXC,KA,2)
  200 CONTINUE
      IF (KKK.EQ.3) RETURN
C-----REDUCTION OF LOAD BLOCKS
  300 NTA = NT(3)
      NTB = NT(4)
      IF (KKK.NE.5) GO TO 310
      NTA = NT(4)
      NTB = NT(3)
```

```
  310 DO 400 M=1,NBLL
C       1. READ LOAD BLOCK M FROM TAPE NTA
        CALL TAPES (NTA,M,A,MAXT,MA,MAXC,KA,1)
C       2. FORWARD REDUCTION OF LOAD BLOCK M
        IF (KKK.EQ.5) GO TO 335
        DO 320 N=1,NBLK
        CALL TAPES(NT(2),N,B,MAXT,MB,MAXC,KB,1)
        CALL BLKOP (A,B,D,MA,MB,KA,KB,NEQ,LEQ,MAXT,MAXC,4)
  320 CONTINUE
C       3. BACKWARD REDUCTION OF LOAD BLOCK M
  330 IF (KKK.EQ.4) GO TO 390
        IF (KKK.EQ.2) GO TO 390
  335 N=NBLK
  340 CALL TAPES(NT(2),N,B,MAXT,MB,MAXC,KB,1)
        CALL BLKOP (A,B,D,MA,MB,KA,KB,NEQ,LEQ,MAXT,MAXC,5)
        N=N-1
        IF (N.GT.0) GO TO 340
C       4. WRITE RESULTS ON TAPE NTB
  390 CALL TAPES(NTB,M,A,MAXT,MA,MAXC,KA,2)
  400 CONTINUE
        RETURN
        END
        SUBROUTINE TAPES (NT,NR,X,MAXT,MC,MAXC,KF,KK)
        COMMON/TAPES/NTAPE(10)
        DIMENSION X(MAXT),MC(MAXC),KF(4)
C       SUBROUTINE TO READ OR WRITE BLOCK OF INFORMATION (X,MC,KF)
C       WHICH IS RECORD  NUMBER NR ON TAPE NT.
C       LDR,LBK SHOULD BE SET PROPORTIONAL TO THE COST OF ONE
C       DUMMY TAPE READ OR ONE TAPE BACKSPACE RESPECTIVELY
C       NTAPE(I) CONTAINS THE CURRENT RECORD POSITION OF TAPE I
C       NTAPE(I) NEED NOT BE INITIALIZED IF THE TAPE WAS WRITTEN
C       BY S/R TAPES OR IF THE FIRST RECORD TO BE READ IS RECORD 1
        LDR=1
        LBK=1
        IF (NR.NE.1) GO TO 90
   50 REWIND NT
        NTAPE(NT) = 1
   90 LR=NR*LDR
        LK=(NTAPE(NT)-NR)*LBK
        IF (LR.LT.LK) GO TO 50
  100 IF (NTAPE(NT)-NR) 200,400,300
  200 READ (NT)
        NTAPE(NT) = NTAPE(NT)+1
        GO TO 100
  300 BACKSPACE NT
        NTAPE(NT) =NTAPE(NT)-1
        GO TO 100
  400 IF (KK.EQ.1) READ (NT) X,MC,KF
        IF (KK.EQ.2) WRITE (NT) X,MC,KF
        NTAPE(NT)=NTAPE(NT)+1
        RETURN
        END
        SUBROUTINE BLKOP (A,B,D,MA,MB,KA,KB,NEQ,LEQ,MAXT,MAXC,KK)
        DIMENSION A(1),B(1),MA(1),MB(1),KA(4),KB(4),D(1)
        GO TO (100,300,300,300,500),KK
C
C       1. MOVE A ARRAY TO B
C
  100 DO 110 I=1,4
  110 KB(I)=KA(I)
        DO 120 I=1,MAXC
  120 MB(I)=MA(I)
```

```
      DO 130 I=1,MAXT
  130 B(I)=A(I)
      RETURN
C
C     2. REDUCE BLOCK A BY BLOCK B
C
  300 NTA=KA(4)-KA(3)+1
      NTB=KB(4)-KB(3)+1
      KJ=0
      DO 390 J=1,NTA
      IF (KK.EQ.4) MA(J)=J*NEQ
      JJ=KA(3)+J-1
      IF (KK.EQ.4) JJ=NEQ
      KHJ=MA(J)-KJ
      IL=NTB
      IF (KK.EQ.3) IL=J
      KI=0
      DO 380 I=1,IL
      II=KB(3)+I-1
      KHI=MB(I)-KI
      KH=MIN0 (KHI,KHJ-JJ+II)
      KF=MB(I)-KH+1
      KL=MB(I)-MAX0 (1,II-LEQ)
      SS=0.0
      IJ=MA(J)-JJ+II
      IF (KK.NE.3) GO TO 310
      IF (I.EQ.J) GO TO 355
  310 IF (KF.GT.KL) GO TO 380
      IC=IJ-MB(I)
      DO 320 K=KF,KL
  320 SS=SS+B(K)*A(K+IC)
      A(IJ)=A(IJ)-SS
      GO TO 380
  355 KD=JJ-KHJ+1-KF
      IF (KF.GT.KL) GO TO 380
C     REDUCE COLUMN BY ITSELF
      DO 370 K=KF,KL
      AD=D(KD+K)
      IF (AD) 360,370,360
  360 T=A(K)/AD
      SS=SS+T*A(K)
      A(K)=T
  370 CONTINUE
      A(IJ)=A(IJ)-SS
  380 KI=MB(I)
      IF (KK.EQ.4) D(II)=B(KI)
      KJ=MA(J)
      IF (KK.EQ.4) GO TO 390
      D(JJ)=A(KJ)
  390 CONTINUE
      IF (KK.EQ.4.AND.KB(4).EQ.NEQ) GO TO 400
      RETURN
C     DIVIDE LOADS BY DIAGONALS
  400 KJ=0
      DO 450 J=1,NTA
      MA(J)=J*NEQ
      KHJ=MA(J)-KJ
      IF=NEQ-KHJ+1
      IF (IF.GT.LEQ) GO TO 450
      DO 440 II=IF,LEQ
      IF (D(II)) 430,440,430
  430 A(II+KJ)=A(II+KJ)/D(II)
```

```
  440 CONTINUE
  450 KJ=MA(J)
      RETURN
C
C     3. BACK-SUBSTITUTION
C
  500 II=KB(4)
      NTA=KA(4)-KA(3)+1
  520 I=II-KB(3)+1
      IF (I.LE.0) RETURN
      KF=1
      IF (I.GT.1) KF=MB(I-1)+1
      KL=MB(I)-MAX0(1,II-LEQ)
      IF (KF.GT.KL) GO TO 560
      KHI=MB(I)-KF+1
      KJ=0
      DO 550 J=1,NTA
      IC=KJ+(II-KHI)-KF+1
      AIJ=A(KJ+II)
      DO 540 K=KF,KL
  540 A(K+IC)=A(K+IC)-B(K)*AIJ
  550 KJ=J*NEQ
  560 II=II-1
      GO TO 520
C
      END
      SUBROUTINE PRAB (A,B,KA,KB,MA,MB)
      DIMENSION A(1),B(1),MA(1),MB(1),KA(4),KB(4)
      DIMENSION RR(14)
      NTA=KA(4)-KA(3)+1
C
      KF=1
      DO 400 I=1,NTA
      NA=KA(3)+I-1
      KL=MA(I)
      DO 320 J=1,NA
  320 RR(J)=0.0
      KH=KL-KF
      L=NA-KH
      DO 350 K=KF,KL
      KK=K-KF
  350 RR(KK+L)=A(K)
      WRITE (6,2000) I,(RR(J),J=1,NA)
  400 KF=KL+1
C
      RETURN
 1000 FORMAT (2H B,I3,14F9.2)
 2000 FORMAT (2H A,I3,14F9.2)
      END
```