REPORT NO.
UCB/SESM-84/06

STRUCTURAL ENGINEERING AND
STRUCTURAL MECHANICS

AN ALGORITHM FOR ASSEMBLY OF STIFFNESS MATRICES INTO A COMPACTED DATA STRUCTURE

by

BAHRAM NOUR-OMID

and

ROBERT L. TAYLOR

MAY 1984

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

AN ALGORITHM FOR ASSEMBLY OF STIFFNESS MATRICES INTO A COMPACTED DATA STRUCTURE

by

Bahram Nour-Omid[†]

and

Robert L. Taylor[‡]

ABSTRACT

A data structure is described that stores only the non-zero terms of the assembled stiffness matrix. This storage scheme results in considerable reduction in memory demand during the assembly phase of a finite element program. Therefore, larger matrices can be formed in the main memory of the computer. When secondary store must be used this approach reduces the I/O cost during the assembly stage.

An algorithm is derived that starts with the element connectivity information and generates the compacted data structure. The element matrices are then assembled to form the stiffness matrix with this storage scheme. The assembly algorithm is described and a FORTRAN listing of the routines are presented. The reduction in storage is demonstrated with the aid of numerical examples.

† Center for Pure and Applied Mathematics, University of California, Berkeley, CA 94563.

‡ Civil Engineering Department, University of California, Berkeley, CA 94563.

This research was supported in part by the Office of Naval Research under contract N00014-76-C-0013 and the Naval Civil Engineering Laboratory under purchase order N62583/83 M T265.

Introduction

The analysis of large structures, especially in three dimensions, can result in stiffness matrices that demand an exceptionally large amount of computer storage. The storage needs of these matrices depend to a large extent on their sparsity and the data structure that is used to store them. The choice of the data structure in turn depends on the method that is used to solve the associated system of equations. Presently, most solution schemes used in finite element computer programs are based on direct methods, i.e. triangular factorization of the stiffness matrix, $\mathbf{K}$. Starting from a given mesh description, a finite element program performs the following steps:

1. determine the sparsity structure of $\mathbf{K}$,
2. renumber the equations to reduce the storage demands of $\mathbf{K}$,
3. reserve the required storage for $\mathbf{K}$,
4. compute the element matrices,
5. assemble the element matrices into $\mathbf{K}$,
6. compute the triangular factorization of $\mathbf{K}$,
7. solve the associate system of equations.

In many applications the available primary memory is not sufficient to store the assembled matrix, and therefore secondary storage is used. In this circumstance, steps 4 through 7 involve data transfers between primary and secondary store, often referred to as I/O. In this case, $\mathbf{K}$ is partitioned into blocks and each block is assembled and stored on secondary store. The blocks are then brought back into the main memory to form the factors of $\mathbf{K}$. For large enough problems the I/O costs can dominate the computation costs.

A great deal of effort has been expended to develop new procedures for reordering the equilibrium equations, thus reducing the overall storage requirements in the solution steps 5, 6 and 7 [1,2,3]. This is motivated by the fact that reduction in storage translates directly into a savings in the I/O costs. Among the many solution schemes used, the frontal method [4] and the profile or skyline method [1] are probably the most popular. In [8] it is shown that when the same nodal elimination ordering is used the profile method performs the same number of operations as the frontal method; in [5] an algorithm is described that delivers a good frontal node ordering for the profile method. The significant difference between these methods is that the frontal method often combines steps 4, 5 and 6. In this way the I/O during the assembly step is overlapped with the I/O in the factorization step; thus, the frontal method results in a saving that is equal to the cost of I/O in step 4. Alternatively, the assembly of a profile stored matrix which is partitioned into blocks requires a multiple pass through the elements to perform the assembly. The principle differences in the I/O costs of the two methods during factorization may be traced to the above differences in the partitioning of the matrix into a frontal or a profile form. The principle disadvantage of the frontal method is the added overhead to retain a small front width during the triangular factorization in step 6 and subsequent resolutions in step 7. For example, in resolution of equations this added overhead may lead to CPU costs which are several times those of a profile stored resolution.

In this paper we take a different approach. We use the following simple observation:

The assembly process is independent of the solution procedure.

In other words, one should use the most efficient data structure for the assembly process, step 5, and then restructure the data for ones favorite solution scheme, i.e., either frontal or profile. In this way one can achieve the same reductions in I/O as the frontal method and at the same time maintain high modularity of the program. Here we develop a data structure that stores only the nonzero terms in the stiffness matrix in a compacted form, and present an algorithm for the assembly of **K** for this storage method. This approach results in considerable reduction in the storage needs during the assembly process. Therefore large matrices often may be fully assembled in-core resulting in a considerable reduction in I/O.

This approach has the added advantage that the program is not built around a single equation solver. One can have many solution procedures by simply expanding the compacted structure of **K** into a form appropriate for each particular solution method. Furthermore, the compacted structure can be used directly for iterative solution techniques such as the conjugate gradient type methods [6,7].

Storage Scheme

We now describe the compacted structure that is used to store **K**. We only consider symmetric matrices, although the extension to the nonsymmetric case is trivial, and store only the upper triangular part of **K** is stored. The diagonal terms of **K** will be stored separately in a single array of length n, where n is the total number of equations. The remaining off-diagonals will be placed in a second array of length r, where r is the total number of nonzero off-diagonal terms in the upper triangular part of **K**. All the elements in the same column will be placed consecutively in this array, starting from the top of the column down to the diagonal (excluding the diagonal term). The columns are stored consecutively from the second to the last. For each entry in this array we store its row number in a corresponding integer array of length r. The example below demonstrates the final storage scheme.

Example:

Consider the matrix

$$\mathbf{K} = \begin{bmatrix} 1 & 9 & & & 12 & & & 15 \\ & 2 & 10 & 11 & & 13 & 14 & \\ & & 3 & & 16 & 17 & & \\ & & & 4 & & & 19 & 21 \\ & & & & 5 & 18 & 20 & \\ & & & & & 6 & & 22 \\ & & & & & & 7 & 23 \\ & & & & & & & 8 \end{bmatrix}$$

The array *diag* contains the diagonal terms of **K** as shown below.

$$diag = [1,2,3,4,5,6,7,8] \; .$$

A real array then stores the off-diagonal terms of **K** and a corresponding integer array denotes the row number of each off-diagonal term as show below.

$$\begin{aligned} off\text{-}diag &= [9,10,11,12,16,13,17,18,14,19,20,15,21,22,23] \\ irow &= [1,\; 2,\; 2,\; 1,\; 3,\; 2,\; 3,\; 5,\; 2,\; 4,\; 5,\; 1,\; 4,\; 6,\; 7] \end{aligned}$$

In addition, a single integer array of length n is also required to point to the end of entries from a given column. For the above example this array is

$$jcol = [0,1,2,3,5,8,11,15]$$

The total storage requirement is $r + n$ real words and $r + n$ integer words. Using a 16 bit integer word, $\frac{5}{4}(n + r)$ real words (64 bit) will be sufficient. Then the largest number of equations that can be solved this way is $2^{15} - 1 \approx 32000$. With a 32 bit integer word, the total storage required will be $\frac{3}{2}(n + r)$ real words.

Derivation of the Assembly Process

In this section we give a step by step derivation of the assembly algorithm. Each step is demonstrated with the aid of the mesh example in Fig. 1. First we introduce some notation. A finite element mesh is denoted by $M = \{E , N\}$ where E and N represent the collection of elements and nodes in the mesh. A part of the input information provided to a finite element program is the set of nodes belonging to an element. This we denote as $N_e \subset N$. For example element 4 in Fig. 1 has the connectivity set $N_4 = \{5,8,9,6\}$. In Table 1 we give the complete list of the connectivity sets N_e for each element in example 1. These data are usually assembled in a single array known as the connectivity array. The complete set $\{ N_e \forall e \in E \}$ is sufficient to describe the connectivity of a given mesh. Another part of the input data is the boundary conditions that determine the set of the indices of all the active degrees of freedom at node p. We denote this set as U_p. In Table 1 we give the set $\{ U_p \forall p \in N\}$ for the example in Fig. 1. In this example, we assume that there are two degree of freedom at each node. The collection of N_e, column 2, and U_p, column 4, given in Table 1 is sufficient to determine the sparsity structure of the stiffness matrix associated with a given mesh.

Element Connectivity		Boundary Conditions	
Element No. e	Set of Nodes for each Element, N_e	Node No. p	Set of Unknowns for the Active nodes, U_p
1	{ 1, 4, 5, 2 }	2	{ 1 }
2	{ 2, 5, 6, 3 }	4	{ 2 }
3	{ 4, 7, 8, 5 }	5	{ 3, 4 }
4	{ 5, 8, 9, 6 }	6	{ 5 }
5	{ 7, 11, 8 }	7	{ 6 }
6	{ 10, 11, 7 }	8	{ 7, 8 }
7	{ 8, 11, 9 }	9	{ 9 }
8	{ 11, 12, 9 }	11	{ 10 }

Table 1. Connectivity sets and active degrees of freedom for the mesh in Fig. 1.

Our objective here is to find the set of indices of the unknowns that are coupled with a given degree of freedom. This is precisely the row number of each nonzero term in a given column of the stiffness matrix, *irow*, that is required for the storage scheme described in the previous section.

First, we must establish the set of elements that are connected to each node. This can be done by inspecting the element connectivity sets. Looking at the second column of Table 1, for example node 4 appears twice, in rows 1 and 3. We then conclude that node 4 is connected to elements 1 and 3. This process must be repeated for each node. The difficulty here is that we do not know *apriori* the number of storage locations needed to identify the set of elements for each node. For this reason the above process is carried out in two steps. The number of elements connected to each node is determined and stored first. We refer to this as the E-degree (element degree) of each node. In the example the E-degree of node 4 is 2. The E-degree also determines the length of the array that is required to keep the set of elements connected to each node. For each node p we denote this set by $E_p \subset E$. Then the above process is simply to evaluate

$$E_p = \{ e \mid p \in N_e \} \tag{1}$$

for each node p. This equation may be thought of as finding the pseudo-inverse of the connectivity array. The E-degree of node p is the number of terms in E_p (the cardinality of E_p). See columns two and three of Table 2 for the E-degree and the complete set of E_p for the nodes in example 1.

Next, we find for the set of nodes that are adjacent to each node p. We denote this by $A_p \subset N$. This is the set of all nodes that belong to an element with p as one of its nodes. Having established the set of elements connected to node p, the adjacent nodes are all the other nodes belonging to these elements. In example 1 node 4 is connected to elements 1 and 3 (see column 2 of Table 2). The set of nodes belonging to elements 1 and 3 are obtained by inspecting column 2 of Table 1; these are { 1, 4, 5, 2 } and { 4, 7, 8, 5 } respectively. Then the set of nodes adjacent to 4 is $A_4 = \{ 1, 5, 2, 7, 8 \}$. The N-degree (nodal degree) of a node is the number of nodes adjacent to it and is the cardinality of A_p. In columns 4 and 5 of Table 2 we give the N-degree and the adjacency set of each node in the nodal graph of example 1 (Fig. 2). This step is simply to evaluate the equation

$$A_p = \bigcup_{e \in E_p} N_e - n \tag{2}$$

Note that both A_p and N-degree can be obtained in the same loop.

Node p	E degree	E_p	N degree	A_p	S degree	S_p
1	1	{ 1 }	3	{ 4, 5, 2 }	0	ϕ
2	2	{ 1, 2 }	5	{ 1, 4, 5, 6, 3 }	1	{ 1 }
3	1	{ 2 }	3	{ 2, 5, 6 }	1	{ 2 }
4	2	{ 1, 3 }	5	{ 1, 2, 5, 7, 8 }	2	{ 1, 2 }
5	4	{ 1, 3, 4, 2 }	8	{ 1, 4, 7, 8, 9, 6, 3, 2 }	4	{ 1, 4, 3, 2 }
6	2	{ 2, 4 }	5	{ 2, 5, 8, 9, 3 }	3	{ 2, 5, 3 }
7	3	{ 3, 5, 6 }	5	{ 10, 11, 8, 5, 4 }	2	{ 4, 5 }
8	4	{ 3, 5, 7, 4 }	6	{ 11, 9, 6, 5, 4, 7 }	4	{ 6, 5, 4, 7 }
9	3	{ 4, 7, 8 }	5	{ 11, 12, 6, 5, 8 }	3	{ 6, 5, 8 }
10	1	{ 6 }	2	{ 7, 11 }	1	{ 7 }
11	4	{ 6, 5, 7, 8 }	5	{ 10, 12, 9, 8, 7 }	4	{ 10, 9, 8, 7 }
12	1	{ 8 }	2	{ 11, 9 }	2	{ 11, 9 }

Table 2. The result of algorithm for establishing the row index of the nonzero terms in **K** for example 1.

Since we want to store only the upper triangular part of **K** we need to store only a subset of A_p. This will be the set of nodes in A_p with an index less than p; that is $S_p = \{i \mid i \in A_p \text{ and } i < p\}$. We refer to the number of terms in S_p as the S-degree (semi-degree) of a node. The set S_p is only useful when the numbering of the unknowns are such that when $i < j$ all the unknowns at node i have a smaller index than the unknowns at node j. Whenever this is not true it is necessary to use the complete set of adjacent nodes A_p together with the numbers of the unknowns for each node (e.g., see listing in Appendix A.).

Finally, for a given unknown at node p with index $j \in U_p$ we find the set of the indices of all other unknowns that are coupled with u_j. This will be the set of row indices $\overline{R}_j$ for nonzeros in j-th column of **K**. Then

$$\overline{R}_j = \bigcup_{\{i \in A_p\}} U_i \tag{3}$$

For example 1 $\overline{R}_j$ is the adjacency set of j in the graph of the unknowns in Fig. 3. Since we only store the upper triangular part of **K** we scan through $\overline{R}_j$ and use the subset defined by:

$$R_j = \{\, i \mid i \in \overline{R}_j \text{ and } i < j \,\} \tag{4}$$

R_j is the row index of all the nonzero terms in the j-th column of the upper triangular part of **K**. The complete set of R_j for the example problem is presented in Table 3.

D.O.F. j	Node p	R_j
1	2	$\{0\}$
2	4	$\{1\}$
3	5	$\{1, 2\}$
4	5	$\{3, 2, 1\}$
5	6	$\{4, 3, 1\}$
6	7	$\{3, 4, 2\}$
7	8	$\{6, 2, 4, 5, 3\}$
8	8	$\{2, 3, 4, 5, 6, 7\}$
9	9	$\{7, 4, 5, 3, 8\}$
10	11	$\{9, 6, 7, 8\}$

Table 3. The row indices of the nonzero terms in the upper triangular part of **K**.

The listing of a FORTRAN program that performs all the steps that is described in this section is given in Appendix A. In this Appendix we also provide the subroutine that uses the row indices to perform the assembly of **K**.

Numerical Result

We use the algorithm described in the previous section to assemble the stiffness matrices of the problems described in Table 6. The total storage required during the assembly step is evaluated. We compare these results to similar results obtained when the assembly is performed directly into a profile data structure. The storage requirement of the compacted assembly is not effected by the node ordering. For the assembly into a profile form, we numbered the nodes across the width of the mesh to reduce the bandwidth of the stiffness matrix. Although, the bandwidth could have been reduced further using a renumbering scheme such as [2,3], we omitted this step for simplicity. The results for examples 2 to 6, given in Table 6, are presented in Table 4 below.

Description of Problem	No. of nodes	No. of elements	No. of equations	Stores for profile	Stores for Compacted **K**
Cantilever type structure	225	184	428	10204	5340
Small Cylinder structure	231	200	440	10492	5679
Large Cylinder structure	496	450	960	32542	12715
$4\times4\times4$ solid structure	125	64	300	21945	11634
$8\times8\times8$ solid structure	729	512	1944	470043	94272

Table 4. Comparison of the storage demands of profile and compacted assembly for **K** in examples 2 to 6.

The results in Table 4 is obtained based on the assumption that a real word is twice as long as an integer word. We observe a reduction from 40% for two dimensional (2-D) problems to 80% for 3-D problems for these examples. The reductions will be more if short integer words are used. It is interesting to note that the required storage for compacted structure varies linearly with the number of equations. Therefore, the saving will be more for larger problems. In Table 5 we give the storage counts for the two methods considered here on square mesh in 2-D and cubes in 3-D as a function of the number of equations. To obtain these estimates we assumed that there is only one degree of freedom per node.

Dimension of Problem	Half bandwidth	Profile storage	Compacted storage
1	2	$5/2\ n$	$3n$
2	$n^{1/2}$	$n^{3/2}$	$15/2\ n$
3	$n^{2/3}$	$n^{5/3}$	$21n$

Table 5. Estimated storage needs for each scheme on regular mesh. n is the number of equations.

Conclusion

The essential steps in a finite element program can be modified to make use of the compacted assembly described here. Accordingly, we perform the following steps:

1. Obtain the row indices of the nonzero terms in **K**.
2. Assemble the matrix in compact form.
3. Choose a solution procedure and renumber the equations to reduce the storage demands of the factors of **K**.
4. expand the compacted **K** into a data structure suitable for the solution method.
5. solve the associated system of equations.

When there is insufficient primary storage, the assembly of the matrix in compacted form opens a number of possible avenues that one can take to reduce the I/O cost. The expansion of the compacted form need not be done immediately after it's assembly. The matrix can be kept in compact form and put on secondary store and expanded into a full profile form only when a factorization must be performed. This way the number of data entries that is read (in the Input phase of I/O) can be reduced considerably, which in turn results in a reduction in the solution time.

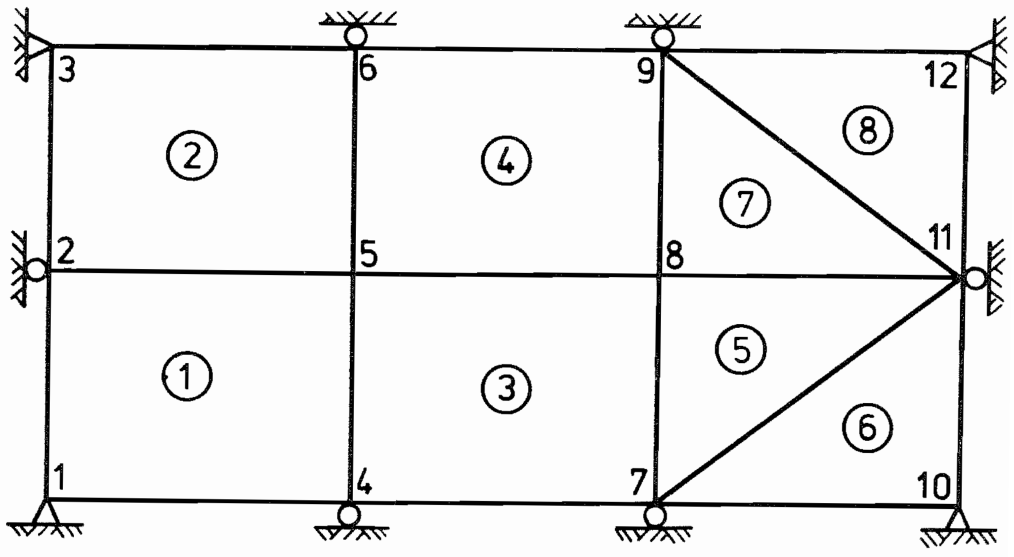


Figure 1. Finite element mesh of example 1.

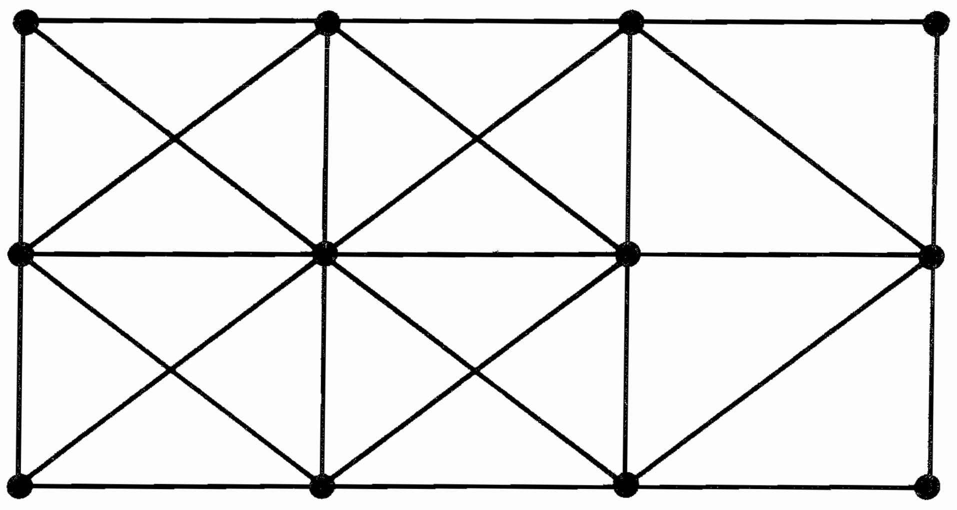


Figure 2. Nodal graph for the mesh of example 1.

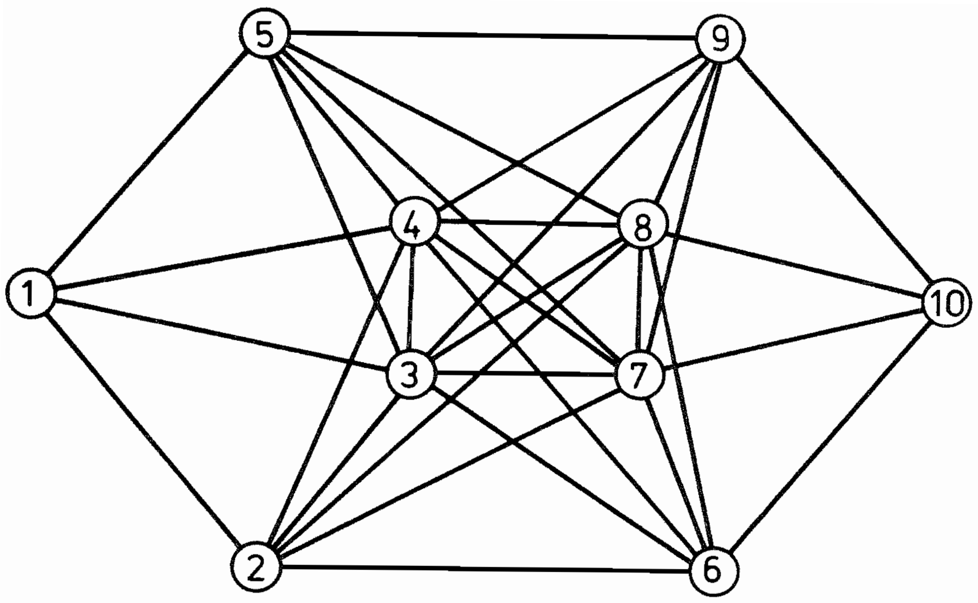


Figure 3. The graph of unknowns for the mesh of example 1.

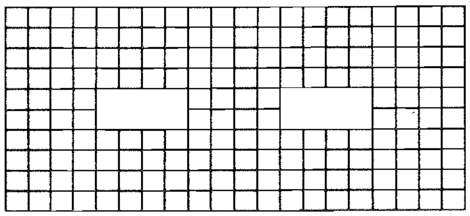
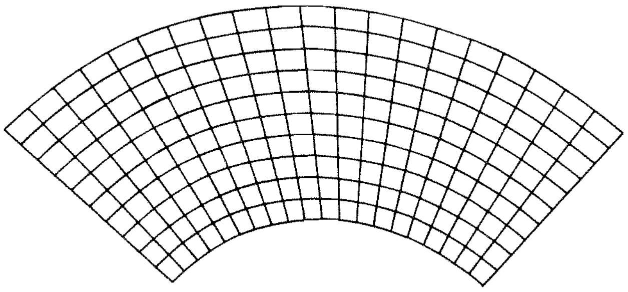
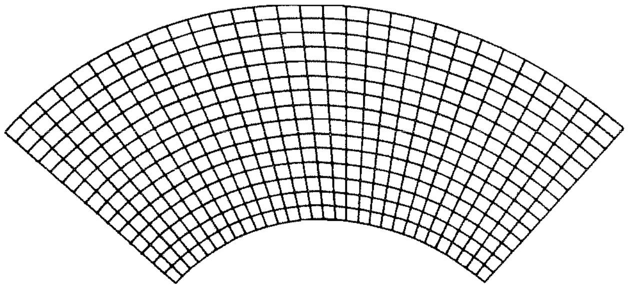
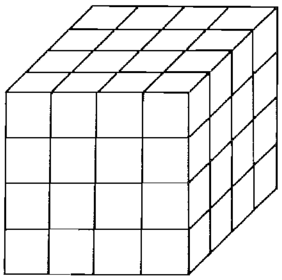
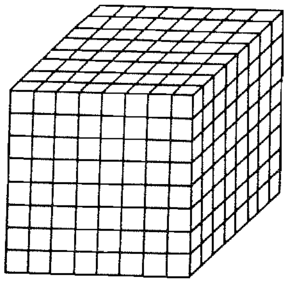
Example 2: Cantilever Structure, left end fixed, plane stress elements with 2 degrees of freedom per node.	
Example 3: Small Cylinder, both ends fixed in tangential direction, plane strain elements with 2 degrees of freedom per node.	
Example 4: Large Cylinder, both ends fixed in tangential direction, plane strain elements with 2 degrees of freedom per node.	
Example 5: 4×4×4 Solid cube, fixed base, solid elements with 3 degrees of freedom per node.	
Example 6: 8×8×8 Solid cube, fixed base, solid elements with 3 degrees of freedom per node.	

Table 6. Description of test examples.

References

[1] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems,* Prentice-Hall, Englewood Cliffs, 1981.

[2] E. Cuthill and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. ACM Nat. Conf.,* New-York, 1969.

[3] N. E. Gibbs, W. G. Poole, Jr., and P. K. Stockmeyer, "An Algorithm for Reducing the Bandwidth and Profile of A Sparse Matrix," *SIAM J. Num. Anal.,* Vol. 13, pp. 236-250, 1976.

[4] B. Irons, "A Frontal Solution Program for Finite Element Analysis," *Int. J. Num. Meth. Engng.,* Vol. 2, pp. 5-32, 1970.

[5] M. Hoit and E. L. Wilson, "An Equation Numbering Algorithm Based on a Minimum Front Criteria," *Computers and Structures,* Vol. 16, No. 1-4, pp. 225-239, 1983.

[6] B. Nour-Omid, B. N. Parlett and R. L. Taylor, "A Newton-Lanczos Method for Solution of Nonlinear Finite Element Equations," *Computers and Structures,* Vol. 16, No. 1-4, pp. 241-252, 1983.

[7] R. L. Taylor and B. Nour-Omid, "Solution of Finite Element Problems by Preconditioned Conjugate Gradient and Lanczos Methods," *Rep. No. UCB/SESM-84/05,* Department of Civil Engineering, University of California, Berkeley, May 1984.

[8] R. L. Taylor, E. L. Wilson and S. Sackett, "Direct Solution of Equations by Frontal and Variable Band, Active Column Methods," in *Nonlinear Finite Element Analysis in Structural Mechanics, Proc. Europe-US Workshop,* July 1976.

[9] R. L. Taylor, "Computer Procedure for Finite Element Analysis," Ch. 24, *The Finite Element Method,* 3-rd Ed., by O. C. Zienkiwicz, McGraw-Hill, London, 1977.

[10] E. L. Wilson, "Solution of Sparse Stiffness Matrices for Structural Systems," *Proc. of Sparse Matrices,* Ed. I. S. Duff, SIAM, Philadelphia, 1979.

Appendix A: Program Listing

```
      SUBROUTINE ELCNT(NUMNP,NUMEL,NEN,NEN1,IX,IC)
      DIMENSION IX(NEN1,1),IC(1)
C
C.... INPUT
C     NUMNP     TOTAL NO. OF NODES IN THE MESH
C     NUMEL     TOTAL NO. OF ELEMENTS IN THE MESH
C     NEN       MAX. NO. OF NODES PER ELEMENT
C     NEN1      DIMENSION OF IX ARRAY
C     IX        ELEMENT CONNECTIVITY ARRAY
C
C.... OUTPUT
C     IC        ARRAY OF LENGTH NUMNP. IT FIRST HOLDS THE ELEMENT DEGREE
C               OF EACH NODE, THEN BECOMES A POINTER FOR AN ARRAY THAT
C               CONTAINS THE SET OF ELEMENTS CONNECTED TO EACH NODE.
C
C.... COUNT THE NUMBER OF ELEMENTS EACH NODE BELONGS TO
C
      CALL IZERO(IC,NUMNP)
      DO 110 N = 1,NUMEL
         DO 100 J = 1,NEN
            I = IX(J,N)
            IF(I.GT.0) IC(I) = IC(I) + 1
100      CONTINUE
110   CONTINUE
C
C.... SET UP POINTERS
C
      DO 120 I = 2,NUMNP
         IC(I) = IC(I) + IC(I-1)
120   CONTINUE
C
      RETURN
      END



      SUBROUTINE CASSEM(D,A,B,S,P,JCOLE,IROW,LD,ID,NST,NEL,AFL,BFL)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      LOGICAL AFL,BFL
      DIMENSION D(1),A(1),B(1),S(NST,1),P(1),JCOLE(1),IROW(1),LD(1)
     1          ,ID(1)
C
C.... COMPACT ASSEMBLY OF PROFILE MATRIX
C
      DO 200 J = 1,NEL
         N = LD(J)
         IF ( AFL .AND. N .GT. 1 ) THEN
            DO 150 I = 1,NEL
               K = LD(I)
               IF ( K .GT. 0 .AND. K .LT. N ) THEN
                  INZ = INZA(JCOLE(N-1)+1,JCOLE(N),IROW,K)
                  A(INZ) = A(INZ) + S(I,J)
               END IF
150         CONTINUE
         END IF
C....    ASSEMBLE THE DIAGONAL
         IF ( N .GE. 1 ) THEN
            IF ( AFL ) D(N) = D(N) + S(J,J)
C....       ASSEMBLE THE LOAD IF NECESSARY
            IF ( BFL ) B(N) = B(N) + P(J)
         END IF
200   CONTINUE
      RETURN
      END
      SUBROUTINE COMPRO(NUMNP,NUMEL,NEN,NEN1,NDF,IX,ID,IC,IROW,IELC,
     1                  JCOLE,KP)
      DIMENSION IX(NEN1,1),ID(NDF,1),IC(1),IROW(1),IELC(1),JCOLE(1)
C
C     FOR (NUMNP,NUMEL,NEN,NEN1,IX,IC) SEE SUBROUTINE ELCNT
C.... INPUT
C     NDF     NUMBER OF UNKNOWNS AT EACH NODE
C     ID      ACTIVE UNKNOWNS AT EACH NODE
C.... OUTPUT
C     IELC    HOLDS THE SET OF ELEMENTS CONNECTED TO EACH NODE
C     IROW    ROW NUMBER OF EACH NONZERO IN THE STIFFNESS MATRIX
C     JCOLE   END OF ENTRIES IN IROW FORM A GIVEN COLUMN
C
C.... FIND ELEMENTS CONNECTED TO NODES
C
      CALL IZERO (IELC,IC(NUMNP))
      DO 230 N = 1,NUMEL
         DO 220 J = 1,NEN
            I = IX(J,N)
            IF ( I .GT. 0 ) THEN
               KP = IC(I)
200            IF ( IELC(KP) .EQ. 0 ) GO TO 210
                  KP = KP - 1
               GO TO 200
210            IELC(KP) = N
            END IF
220      CONTINUE
230   CONTINUE
C
C.... SET UP COMPRESSED PROFILE POINTERS
C
      KP = 0
      NEP = 1
      DO 350 I = 1,NUMNP
         NE = IC(I)
         DO 340 II = 1,NDF
            NEQ = ID(II,I)
            IF ( NEQ .GT. 0 ) THEN
               JCOLE(NEQ) = KP
               KPO = KP + 1
               IF ( NEP .LE. NE ) THEN
                  DO 330 N = NEP,NE
                     NN = IELC(N)
                     DO 320 J = 1,NEN
                        K = IX(J,NN)
                        DO 310 JJ = 1,NDF
                           NEQJ = ID(JJ,K)
                           IF (NEQJ .GE. NEQ .OR. NEQJ .LT. 0) GO TO 310
C
C....                      CHECK TO SEE IF NODE ALREADY IN LIST
C
                           IF ( KPO .LE. KP ) THEN
                              DO 300 KK = KPO,KP
                                 IF( IROW(KK) .EQ. NEQJ ) GO TO 310
300                           CONTINUE
                           END IF
                           KP = KP + 1
                           IROW(KP) = NEQJ
310                     CONTINUE
320                  CONTINUE
330               CONTINUE
                  JCOLE(NEQ) = KP
               END IF
            END IF
340      CONTINUE
         NEP = NE + 1
350   CONTINUE
      RETURN
      END
      INTEGER FUNCTION INZA(N1,N2,IROW,K)
      DIMENSION IROW(1)
C
C.... FIND THE TERM FOR THE ASSEMBLY
C
      DO 100 N = N1,N2
         IF ( IROW(N) .EQ. K ) THEN
            INZA = N
            RETURN
         END IF
100   CONTINUE
C.... ERROR IF LOOP EXITS
      STOP
      END



      SUBROUTINE IZERO(IA,NN)
      DIMENSION IA(NN)
      DO 100 N = 1,NN
         IA(N) = 0
100   CONTINUE
      RETURN
      END
```