INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 20, 625–641 (1984)

SOLUTION OF LINEAR EQUATIONS FOR SMALL COMPUTER SYSTEMS*

NATHAN IDA AND WILLIAM LORD

Department of Electrical Engineering, Colorado State University, Fort Collins, Colorado, U.S.A.

SUMMARY

A solution technique, based on Gauss elimination, is described which can solve symmetric or unsymmetric matrices on computers with small core and disk requirement capabilities. The method is related to frontal techniques in that renumbering of the nodes, such as in a finite element mesh, is not required, and the elimination is performed immediately after the equations for a particular node have been fully summed. Only two rows of the matrix need be on core at any step of the solution, but for more efficiency, the program presented here requires all the equations associated with two nodes to be on core. Minimum disk storage is achieved by storing only nonzero entries of the matrix, a single pointing vector for each node, regardless of the number of degrees-of-freedom, and the use of a single sequential file. Special care is taken of the boundary nodes where only the diagonal and the right-hand-side vector are stored. Assembly and elimination for these nodes are avoided completely. The performance of the program is compared with both symmetric and nonsymmetric frontal routines and is shown to be acceptable. The major merit of the method lies in the fact that it can be implemented on small minicomputers. The reduction of core and disk storage inevitably increases the solution time, but the decrease in the output file size also makes the back-substitution and resolution processes more efficient. In some cases, the total solution time can be shorter than for the frontal method due to this property.

INTRODUCTION

The need to solve large systems of equations has led to two well-established approaches, the banded and frontal methods. The various frontal methods[1–3] were developed to accommodate large matrices with relatively small core requirements on medium to large size computers, whereas the banded solution methods[4] rely mainly on availability of large core on the larger computers. The first approach makes efficient use of large and fast disks to achieve reasonable solution times and can be said to be I/O intensive. The second approach makes use of fast CPUs to work within a large core or more likely within a large virtual memory. This can be said to be CPU intensive, although the use of virtual memory tends to blur this distinction.

There is, however, a third approach which is becoming more popular and more feasible. It is that of solution of medium to large sized problems on small minicomputers.[5] This approach seems at first to have the basic disadvantages of the previous two approaches, namely, smaller and slower disks and less powerful CPUs. Programming solution algorithms on this class of computers is not merely a tradeoff between core and solution time, but rather, the practicability of the process must be established first and then both the CPU usage and I/O processes optimized to the point where the solution algorithm becomes a useful alternative in terms of computer time and cost. The fact that usage of minicomputers is available for long periods of time with little or no running costs makes this alternative a very attractive one.

The solution method presented here, based on the Gauss elimination algorithm, was designed to use a minimum of core and to minimize at the same time the disk storage requirements

* This work was performed with the support of the Army Research Office and the Electric Power Research Institute.

0029–5981/84/040625–17$01.70

Received 15 November 1982
Revised 3 January 1983

while still remaining feasible in terms of solution time. Realizing that the most expensive and slower operation, one which is extensively used in both the frontal and the present method, is the back-space operation, special care was taken to shorten the records written in the output file during elimination and to minimize the number of records. Thus, for example, a record is written or read for each node rather than for each variable.

The algorithm presented here was implemented to run on a Tektronix 4081 graphics system with about 28 Kbyte of available core and 2·5 mbyte of disk storage. Problems having in excess of 2000 variables were successfully solved, but for the sake of generality the program listed in Appendix II is written to run on a VAX 11/780.

Results of a medium sized three-dimensional problem are presented and compared with solutions obtained with the symmetric frontal method of Irons[1] and the nonsymmetric solution of Hood.[2] This comparison is also done on a VAX 11/780 computer since none of the frontal methods could be run on the Tektronix minicomputer for any realistic sized problem.

THE PROGRAM'S PRINCIPLES

The solution process follows the basic Gauss algorithm as used in many banded methods, but relies on an assembly process which is close to that of frontal methods. The assembly is done node by node rather than the common assembly by elements. Beginning with the first node in the mesh, the files containing the mesh data are scanned and all the necessary elements assembled. At the end of this process the equations associated with the current node are fully summed and ready for elimination. As part of the assembly process a pointing vector is created to identify each variable in the currently assembled equations. These equations are next eliminated and only then is the next node assembled. This process requires partial assembly of each element as many times as there are nodes in the element, but it permits the assembly of the current node without storing any data relating to any of the other nodes in the mesh.

The equations needed for the elimination of the current node are not present on core: therefore, the output file is searched for the necessary equations and these are loaded and used in the sequence they are detected. Each output record contains all the equations of one node and these are all loaded and used in a single elimination step.

To ensure that the output file will not have to be searched more than once for each node being eliminated, the pointing vector for the equations in the current node are arranged in ascending order of variables and the values in the corresponding locations in the equations interchanged accordingly. This guarantees that at any stage of the elimination process, the variables to be eliminated are always at the beginning of the equations. After all the variables preceding the diagonal have been eliminated, the elimination proceeds with the variables of the current node, and each equation is normalized with respect to its diagonal. The output file is advanced to the end-of-file and the normalized equations written on file along with their pointing vector in a single record.

The back-substitution process is similar to that used in many frontal routines. The output file is back-spaced, one record is read and the file back-spaced again. Each record, however, contains all the equations associated with a single node and, therefore, the variables of this node are evaluated before the next step takes place.

The resolution facility is not a separate one as is customary in frontal methods. Rather, all the right-hand-side vectors are created during assembly, modified during elimination and written in the output record. The resolution then is merely a repetition of the back-substitution process with a new right-hand-side vector, after the output file has been advanced to the end-of-file.

Programming details

The program consists of the main program and one subroutine. Program RSOL performs the assembly, elimination, back-substitution and resolution, whereas subroutine SORT rearranges the pointing vector and the equations in ascending variable numbers. A third subroutine, referred to as ELEMR in the program, is needed for the assembly of the elemental matrices, and has to be supplied by the user as it is problem dependent. The program is listed in Appendix II, whereas a definition of variables is provided in Appendix I.

In discussing the program details, the 9-element, 16-node mesh in Figure 1 is considered. It is assumed that there are three degrees-of-freedom at each node and two right-hand-side vectors. In each element the node numbering is assumed counterclockwise, starting with the lower left node.

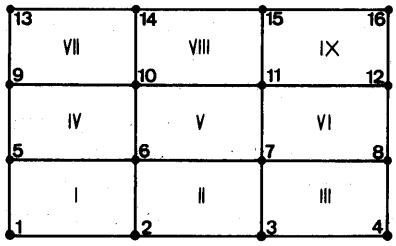

Figure 1. A 16-node mesh

Assembly

The assembly starts with node number 1 and proceeds node by node. This method is adopted since it is assumed that only the equations associated with two nodes can be present on core at any time. The normal way of frontal assembly cannot be used. In addition, the mesh data such as connectivity and co-ordinate arrays as well as boundary conditions are assumed to be in files on disk. Any search for data needed for assembly is done on disk, although a much faster assembly could be performed if these data were on core.

For the current node being assembled, the elements containing this node need to be formulated and their contribution summed. Thus, for example, for node number 6, elements I, II, IV and V are the only elements contributing to the equations associated with this node. Moreover, since these are the only equations that can be summed, only the rows in the elemental matrix corresponding to the current node in each element need be formulated. For node number 6, element number 1 is first assembled for which rows 7, 8 and 9 in the elemental matrix are created in RW and returned for summing in ROW. The contribution of this element to the pointing vector is created in LDES and updated in LDEST. The right-hand-side values are also created by subroutine ELEMR and returned as the last entries in RW. For the mesh in Figure 1, RW will always contain 12 entries for each of the 3 variables and 2 values for the right-hand-side in locations 13 and 14. The length of the equations after assembly is equal to the number of variables associated with one node plus the number of right-hand-side vectors. The maximum length in this example is 29. Initially, as the variables are summed in LDEST and ROW these are assumed to be of this length such that the right-hand-side contributions are summed into locations 28 and 29. The situation in LDEST is shown in Figure 2(a) after

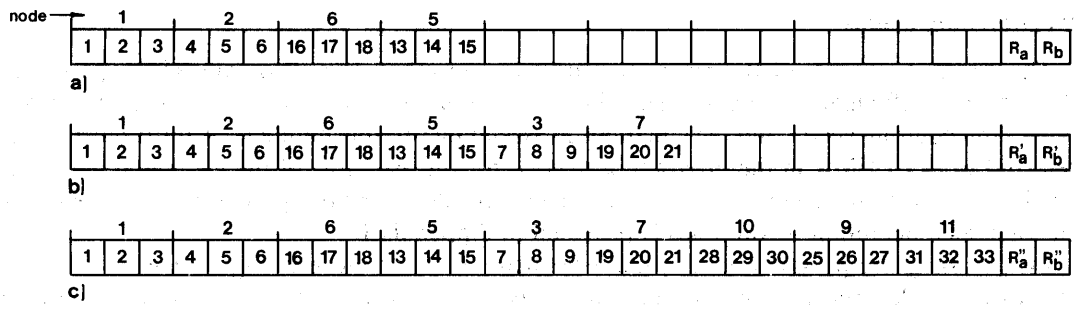

Figure 2. Array LDEST during assembly

element I is assembled. When assembling element number II, rows 10, 11 and 12 in the elemental matrix are created and summed into ROW. The pointing vector is updated as in Figure 2(b). After assembly of elements IV and V the process is complete and the status of the pointing vector is as shown in Figure 2(c). At this stage the variables in LDEST and ROW are reordered in ascending order of variables and the right-hand values are moved to the left such that no zero entries exist. This would be the case for node number 1 where only element number 1 is contributing and the total length in ROW and LDEST is 14. The equations of node 6 are now fully summed and are eliminated, before assembly of node number 7 can begin.

Elimination

Elimination begins by rewinding the output file and reading the first record into LDES and RW. The first location in LDES is compared with the first location in LDEST. If they match, this record is needed to eliminate the first NDF variables in the current equations. If these values do not match, the next record is read and the process repeated until all the variables preceding the current variables have been eliminated. Taking node 5 as an example, the equations associated with node 1 are read as the first record. Variables 1, 2 and 3 are eliminated and the remaining variables shifted to the left by calling subroutine SORT. This subroutine also rearranges the variables in ascending order such that the next variables to be eliminated will be located in the first three locations in LDEST, as in Figure 3(c). The next record is read and variables 4, 5 and 6 eliminated. This record contains variables 7, 8 and 9 from node number 3 and 19, 20 and 21 from node number 7 which were not present in equations 13, 14 and 15 associated with node number 5. These are inserted between the last variable of the equations and the right-hand values by shifting the latter six locations to the right, resulting in the situation in Figure 3(d). SORT is again called and the equations rearranged as in Figure 3(e). This process is repeated until variables 13, 14 and 15 appear in the first three locations in LDEST. At this point, internal elimination within these three equations takes place whereby variable 13 is eliminated from equations 14 and 15 and variable number 14 eliminated from equation 15. This situation is summarized in Figure 3(f). In this case, the final length of LDEST and ROW has not changed but, in general, it can be longer or shorter than the length before elimination (see section on dimensioning).

Each of the NDF equations is now normalized by dividing it by its diagonal value and written in the output file after it has been advanced to the end-of-file.

It should be noted that the right-hand-side vectors are updated at each step, making them ready for back-substitution and resolution.

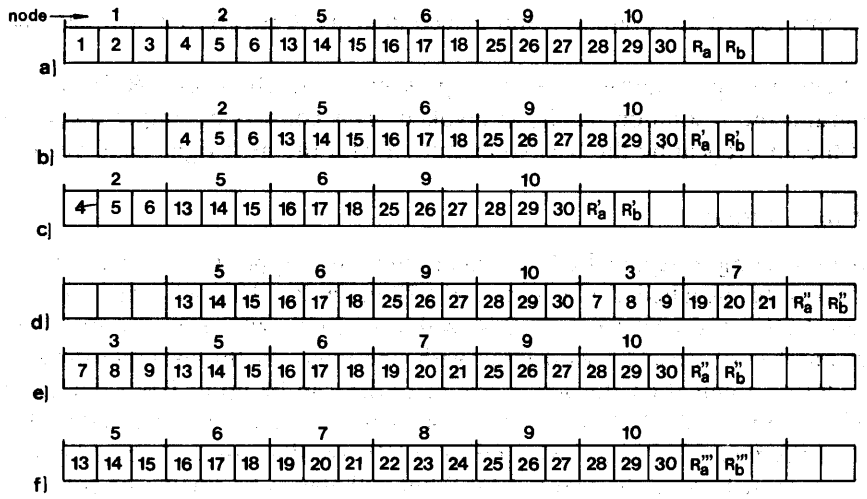

Figure 3. Array LDEST during elimination

Boundary conditions

The assembly and elimination processes are skipped for any node which has specified boundary conditions. These nodes are identified as part of the input in NBOUND.

For each boundary node, the first location in LDEST is assigned the highest variable number associated with this node. The second location, corresponding with the first right-hand-side value, is encoded with the value 1 to distinguish it as a boundary equation. The first location of each of the NDF rows in ROW is assigned a value of 1·0 and the corresponding boundary conditions inserted in the right-hand vector locations. Thus, for example, if node 12 is a boundary node the situation in LDEST and ROW will be as in Figure 4, assuming that all boundary conditions are equal to 0·0 for the first right-hand-side vector and to $1{\cdot}0E+06$ for the second.

These equations are written directly to the output file for further use in elimination and back-substitution. The boundary node records are very short contributing to the overall small size of the output file.

Back-substitution and resolution

The back-substitution process is straightforward: the output file is back-spaced, a record is read into LDEST and RW and then the output file back-spaced again. At this stage all NDF

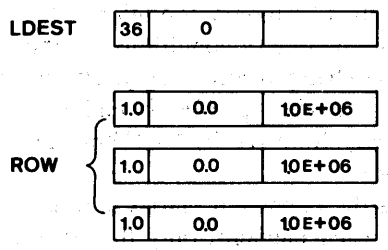

Figure 4. Arrays LDEST and ROW for boundary nodes

variables in this record are evaluated and placed in array A before the next back-substitution step begins.

If more than one right-hand-side vector is present, as initialized by NRHS at the beginning of the program, the resolution facility is invoked. This consists of simply reading the file to the end-of-file and repeating the back-substitution with the next right-hand vector.

Dimensioning of arrays

The program listing in Appendix II is dimensioned for a typical three-dimensional problem with 200 first-order hexahedral elements and 972 variables. The only arrays that need some attention are LDES and LDEST. LDES is used in assembly where its size should be equal to the largest number of variables associated with one node. In the example presented above, this is 29. During elimination the active size of LDES can be up to BW+NRHS, where BW is the semibandwidth and NRHS the number of right-hand-side vectors or 20 in this example. The larger of the two should be used. LDE and RW are dimensioned accordingly.

LDEST can grow beyond the size of LDES but will always be smaller or equal to 2BW-1+NRHS. In most cases a size which is about 1·5 times that of LDES will suffice, as is the case in the listing in Appendix II. ROW is dimensioned according to LDEST.

INPUT OF DATA AND SUBROUTINE ELEMR

The input data needed for the solution consists of element data, node co-ordinates and boundary conditions. The elements data is stored in file NODARRAY.DAT. The first entry in each record of this file is NNPE, followed by the node numbers of the current element. The last entry is MAT which assigns a material identification number to each element. This is later used in ELEMR to introduce the material properties in the elemental matrix assembly.

Before ELEMR can be called to create the elemental matrix, it is necessary to read in the co-ordinates of the nodes of the current element from file NODPNT.DAT. Each record in this file contains the x, y and z co-ordinates of a node and the records arranged in ascending node numbers. The file is scanned and the co-ordinates of the NNPE nodes of the current element are read in array COOR. These are identified by comparing each node of the element with the record number.

The boundary conditions data is contained in two files. NBOUND.DAT contains first (in a single record) the node numbers that have specified boundary conditions. These are read and stored on core in array NBOUND. Next, this file contains the number of variables per node (NDF) for each node in the mesh. Each value of NDF is written as a separate record and read in for the current node (NEP) before assembly begins. If all the nodes in the mesh have the same number of degrees-of-freedom, it is more efficient to initialize NDF at the beginning of the program. Later, after the present equations are fully summed, array NBOUND is scanned to check if the node has prescribed boundary conditions. If so, file BCOND.DAT is searched for the corresponding node, the boundary conditions for the NDF equations associated with this node read and the equations changed accordingly. This file contains a record for each node that is identified in NBOUND as having specified boundary conditions. The first entry in each record is the node number (KKI), followed by NDF boundary conditions which are stored in array BCX for the current node.

Subroutine ELEMR creates the elemental matrix and has to be supplied by the user. It is somewhat unique in that, for each element, only the rows corresponding to one node are transferred back to RSOL and therefore only part of the matrix need be assembled. For each

node being assembled, ELEMR is called as many times as the number of elements containing this node. In each call, the rows in the elemental matrix corresponding to the current node are assembled and then transferred back to RSOL in array RW and summed into array ROW. This procedure has been detailed in the section on assembly. The rows that need assembly are identified by KEL. This subroutine also needs to supply the material properties, perhaps in a DATA statement. If an existing subroutine is to be used, it is perhaps simpler to create the whole elemental matrix in each call to ELEMR but to transfer back only the NDF rows defined by KEL, (i.e. rows NDF*KEL, NDF*KEL-1, ... NDF*KEL−NDF+1). All data necessary for the elemental matrix assembly is transferred through COMMON/C2/.

COMPARISON WITH FRONTAL ROUTINES

The method presented in the previous sections was compared with both the symmetric frontal solution of Irons[1] and the nonsymmetric frontal routine of Hood.[2] This latter routine was used without pivoting to render the comparison valid. Although the main advantage of the current method lies in the possibility of running medium to large size programs on small core small disk minicomputers, it was practically impossible to compare the three programs on such a computer. The core requirements, especially for the nonsymmetric frontal routine, were too large for a realistic problem to be implemented. On the other hand, a very small test problem would not reveal the limitations or the advantages of this method. The comparison was therefore done on a VAX 11/780 computer. The program listing in Appendix II is also the VAX 11/780 version used for the comparison. The method was, however, implemented on a Tektronix 4081 graphics system and used to run three-dimensional finite element solutions in excess of 2000 variables in size. Some data on this implementation is also provided in addition to the mentioned comparison.

The comparison for the nonsymmetric case is done on a three-dimensional magnetostatic problem solving for the magnetic field around a slot in a current carrying steel bar. The three components of the Magnetic Vector Potential are calculated at each node of the mesh. The mesh has 324 nodes (972 variables) and 200 first-order hexahedral elements.

The symmetric case is compared by solving a three-dimensional heat conduction problem for the temperature at each node of the above mesh.

Table I shows the data for the nonsymmetric case. The time needed for assembly and elimination is much higher than in the frontal routine, but that needed for back-substitution or resolution is significantly lower. Indeed, the total time for the solution is shorter than for the frontal method. As the number of variables per node and the number of right-hand-side

Table I. Comparison of results for nonsymmetric routines

	Current routine		Frontal routine	
Assembly	245	sec.	295	sec.
Elimination	1266	sec.		
Back-substitution	189	sec.	1565	sec.
Total solution time	1511	sec.	1760	sec.
Resolution	195	sec.	1603	sec.
Total disk storage	367616	bytes	1989632	bytes
Disk storage/variable	604·18	bytes	2046·95	bytes
Total core	2129	words	55588	words

vectors increase, the performance of the current routine should improve in comparison to the frontal routine.

More importantly, from the point of view of the routine's merit, as stated in the introduction, is the significant reduction of core and disk storage. The disk storage is reduced by a factor of 5·4 and the core storage by a factor of more than 26.

Table II summarizes the data for the symmetrical case. In this case, only one variable per node is used. The assembly and elimination times are again longer, but the back-substitution times for both routines are comparable. The reason for the reduced back-substitution time is that the symmetric frontal routine uses back-substitution by elements, whereas the current routine back-substitutes node by node. There are fewer records in the frontal routine output file although they are much longer, as suggested by the disk storage data in Table II. In contrast, the nonsymmetric frontal routine uses back-substitution by variables, accounting for the large back-substitution time.

Table II. Comparison of results for symmetric routines

	Current routine		Frontal routine	
Assembly	220	sec.	11	sec.
Elimination	137	sec.	6	sec.
Back-substitution	65	sec.	46	sec.
Total solution time	422	sec.	63	sec.
Resolution	72	sec.	98	sec.
Total disk storage	65536	bytes	164452	bytes
Disk storage/variable	328·69	bytes	507·26	bytes
Total core	731	words	4536	words

Again, as for the nonsymmetric solution, the routines merit should be judged mainly by the reduction in core and disk storage. The disk storage is reduced by a factor of 6·2, whereas the core requirements are reduced by a factor of 2·5.

Table III summarizes the data for both the symmetric and nonsymmetric solutions on a Tektronix 4081 graphics minicomputer. The time measurements are in total time rather than CPU time used in Tables I and II. The word length used is 32 bit, as for the VAX 11/780.

Table III. Solution on the Tektronix 4081 graphics system

	Symmetric routine	Nonsymmetric routine
Total solution time	2 hrs. 12 min.	8 hrs. 37 min.
Total disk storage	65536 bytes	367616 bytes
Total core	731 words	2129 words

CONCLUSIONS

A solution method has been described which can solve for symmetric and nonsymmetric systems of equations with significant reduction in core and disk space requirements. This accounts for the practical implementation of the algorithm on a small minicomputer, where neither a

symmetric nor a nonsymmetric frontal routine can be implemented. The large solution times are, in many cases, more than compensated for by the low running costs and availability of such minicomputers. The program can also be used as a test program before very large and expensive programs are run on large computers.

APPENDIX I: DEFINITION OF VARIABLES

A(NUMVAR) = Array, stores the results during back-substitution.
BCX(NDF) = Stores the boundary conditions for the current node, if any.
COOR(3, NNPE) = Co-ordinate array for the current element being assembled.
KTEST = Variable transferred to SORT to indicate if reordering is needed or only shift to the left.
KEL = Transfers the current node value to ELEMR. This variable then governs the rows in the elemental matrix that are assembled.
KKI = Current boundary node number being read.
LDES(*) = Pointing vector: used in assembly to create the pointing vector and in elimination to store the pointing vector for the equations read from file.
LDEST(*) = During elimination stores the pointing vector for the node being eliminated. Also used for back-substitution.
LDE(*) = A temporary storage for LDES during elimination.
MAT = Material property variable. The value of this variable refers to the material properties assigned to each element in ELEMR.
MAXN = A variable assigned a value larger than the number of variables in the mesh. Used to detect the length of LDEST.
NBA = NBA points to the location of the first right-hand vector.
NBB = NBB points to the location of the last variable in each equation being eliminated.
NBOUND( ) = Array, stores the node numbers which have boundary conditions (applied).
NDF = Number of degrees-of-freedom per node.
NEP = Current equations (node) in the elimination process.
NH = Number of nodes having specified boundary conditions.
NNPE = Number of nodes per element.
NODV = During elimination it has the value of the first variable in the current node. It also stores the value of NUMVAR for subsequent resolution.
NPT(NNPE) = Node number array for the current element being assembled.
NRHS = Number of right-hand-side vectors.
NUMEL = Number of elements.
NUMNP = Number of nodes.
NUMVAR = Number of variables. During elimination this counts the number of variables eliminated. Also used in back-substitution to sense the end of the process.
RA(NDF) = Temporary storage for one variable in each of the NDF rows in ROW.
RW(NDF, *) = During assembly this contains the rows assembled in ELEMR. During elimination it contains the rows read from file, and during back-substitution it contains the rows read from file for variable calculation.

* See section on dimensioning of arrays.

RWA(NDF, NRHS) = Temporary storage for the right-hand-side vector during elimination when the length of ROW and LDEST are increased.

ROW(NDF, *) = Contains the equation being eliminated during elimination.

APPENDIX II: PROGRAM LISTING

```
      COMMON/C1/LDEST(177),ROW(3,177),K,KTEST,NDF,MAXN,NRHS
      COMMON/C2/RW(3,135),COOR(3,8),NPT(8),MAT,NNPE,KEL
      DIMENSION RWA(3),RA(3,3),LDES(135),LDE(135)
     1,NBOUND(150),BCX(3),A(972)
      EQUIVALENCE (A(1),LDES(1)),(A(136),LDE(1))
     1,(A(271),NBOUND(1))
C
C     VARIABLE INITIALIZATION AND INPUT
C
      NEP=0
      NRHS=1
      NBA=81
      NBB=NBA-1
      NUMEL=200
      MAXN=99999
      NUMNP=324
      NUMVAR=0
C
      OPEN(UNIT=1,NAME='TEMPOR.DAT',FORM='UNFORMATTED',TYPE='NEW')
      OPEN(UNIT=2,NAME='NODARRAY.DAT',FORM='UNFORMATTED',TYPE='OLD')
      OPEN(UNIT=3,NAME='BCOND.DAT',FORM='UNFORMATTED',TYPE='OLD')
      OPEN(UNIT=4,NAME='NBOUND.DAT',FORM='UNFORMATTED',TYPE='OLD')
      OPEN(UNIT=5,NAME='NODPNT.DAT',FORM='UNFORMATTED',TYPE='OLD')
      OPEN(UNIT=8,NAME='OUTP.DAT',FORM='FORMATTED',TYPE='NEW')
C
      READ(4)NH,(NBOUND(KL),KL=1,NH)
      WRITE(6,700)
      WRITE(6,701)(NBOUND(LK),LK=1,NH)
      WRITE(6,702)
      WRITE(6,709)
C
C     ASSEMBLY
C
   99 CONTINUE
      NEP=NEP+1
      IF(NEP.GT.NUMNP)GO TO 199
      READ(4)NDF
      DO 60 JJ=1,NH
      IF(NBOUND(JJ).EQ.NEP)GO TO 64
   60 CONTINUE
      KK=0
      KN=NBA+NRHS
      DO 12 IJ=1,KN
      LDEST(IJ)=MAXN
      DO 12 KJ=1,NDF
   12 ROW(KJ,IJ)=0.
      REWIND 2
C
      DO 14 J=1,NUMEL
      READ(2)NNPE,(NPT(KL),KL=1,NNPE),MAT
C     WRITE(6,703)NNPE,(NPT(KL),KL=1,NNPE),MAT
```

* See section on dimensioning of arrays.

```
      DO 16 KEL=1,NNPE
      IF(NPT(KEL).NE.NEP)GO TO 16
      GO TO 18
   16 CONTINUE
      GO TO 14
   18 CONTINUE
      KM=KEL
      KK=KK+1
      KJP=1
      REWIND 5
      DO 22 II=1,NUMNP
      DO 24 KP=1,NNPE
      IF(II.NE.NPT(KP))GO TO 24
      GO TO 26
   24 CONTINUE
      GO TO 30
   26 READ(5)(COOR(IK,KP),IK=1,3)
C     WRITE(6,704)(COOR(IK,KP),IK=1,3)
      DO 28 KJ=1,NDF
   28 LDES(NDF*KP-KJ+1)=NDF*NPT(KP)-KJ+1
      IF(KJP.EQ.NNPE)GO TO 32
      KJP=KJP+1
      GO TO 22
   30 READ(5)
   22 CONTINUE
   32 CONTINUE
C
      CALL ELEMR
C
      MM=NDF*NNPE
      IF(KK.NE.1)GO TO 42
      DO 34 KJ=1,NDF
      DO 34 JJ=1,MM
      ROW(KJ,JJ)=RW(KJ,JJ)
   34 CONTINUE
      DO 38 JJ=1,MM
   38 LDEST(JJ)=LDES(JJ)
      DO 40 IJ=1,NRHS
      DO 40 JJ=1,NDF
   40 ROW(JJ,NBA+IJ)=RW(JJ,MM+IJ)
      LDEST(NBA+1)=LDES(MM+1)
      GO TO 14
   42 DO 44 M=1,MM
      DO 44 LM=1,NBA
      IF(LDES(M).EQ.MAXN)GO TO 44
      IF(LDES(M).EQ.LDEST(LM))GO TO 46
      IF(LDEST(LM).NE.MAXN)GO TO 44
      LDEST(LM)=LDES(M)
      LDES(M)=MAXN
      GO TO 48
   46 LDES(M)=MAXN
   48 DO 50 KJ=1,NDF
   50 ROW(KJ,LM)=ROW(KJ,LM)+RW(KJ,M)
   44 CONTINUE
      DO 52 IJ=1,NRHS
      DO 52 KJ=1,NDF
   52 ROW(KJ,NBA+IJ)=ROW(KJ,NBA+IJ)+RW(KJ,MM+IJ)
   14 CONTINUE
   20 CONTINUE
      DO 54 II=1,NBB
      IN=II+1
```

```
      DO 56 JJ=IN,NBA
      IF(LDEST(JJ).EQ.0)GO TO 54
      IF(LDEST(II).LT.LDEST(JJ))GO TO 56
      KG=LDEST(II)
      LDEST(II)=LDEST(JJ)
      LDEST(JJ)=KG
      DO 58 JK=1,NDF
      RWW=ROW(JK,II)
      ROW(JK,II)=ROW(JK,JJ)
      ROW(JK,JJ)=RWW
   58 CONTINUE
   56 CONTINUE
   54 CONTINUE
      GO TO 62
   64 CONTINUE
      REWIND 3
      DO 66 JJ=1,NH
      READ(3)KKI,(BCX(JK),JK=1,NDF)
C     WRITE(6,705)KKI,NH,(BCX(JK),JK=1,NDF)
      IF(KKI.EQ.NEP)GO TO 68
   66 CONTINUE
   68 CONTINUE
      KJ=NEP*NDF
      LDEST(1)=KJ
      LDEST(NBA+1)=1
      DO 70 JK=1,NDF
      ROW(JK,1)=1.0
      DO 70 IJ=1,NRHS
      ROW(JK,NBA+IJ)=BCX(JK)
   70 CONTINUE
      KK=2
      LDEST(2)=1
      DO 72 IJ=1,NRHS
      DO 72 KL=1,NDF
   72 ROW(KL,IJ+1)=ROW(KL,NBA+IJ)
      GO TO 80
   62 CONTINUE
      KK=0
      DO 76 KI=1,NBA
      IF(LDEST(KI).EQ.MAXN)GO TO 74
      KK=KK+1
   76 CONTINUE
   74 CONTINUE
      KK=KK+1
      LDEST(KK)=LDEST(NBA+1)
      DO 78 IJ=1,NRHS
      DO 78 KL=1,NDF
   78 ROW(KL,KK+IJ-1)=ROW(KL,NBA+IJ)
   80 CONTINUE
C     NKK=KK+NRHS-1
C     DO 88 KJ=1,NDF
C     WRITE(6,706)(ROW(KJ,LJ),LJ=1,NKK)
C  88 CONTINUE
C     WRITE(6,707)(LDEST(LJ),LJ=1,NKK)
c     WRITE(6,708)NEP
C
C     ELIMINATION
C
      NUMVAR=NUMVAR+NDF
      MK1=KK-1
```

```
      K=KK
      IF(LDEST(KK).NE.1)GO TO 110
      KK=KK+NRHS-1
      WRITE(1)NDF,KK,(LDEST(I),I=1,KK),((ROW(JH,I),I=1,KK),
     1JH=1,NDF)
      GO TO 99
  110 CONTINUE
      REWIND 1
      K1=0
      NODV=NUMVAR-NDF+1
      LL=0
  112 KK=LDEST(1)
      LL=LL+1
      IF(KK.GE.NODV)GO TO 150
  114 READ(1)NDF,MK,(LDES(II),II=1,MK),((RW(JH,II),II=1,MK),JH
     1=1,NDF)
      MK=MK-NRHS+1
      K1=K1+1
      LD=LDES(1)
      IF(LDES(MK).EQ.1)LD=LD-NDF+1
      IF(LD.NE.LDEST(1))GO TO 114
      IF(LDES(MK).EQ.1)GO TO 146
      MK2=MK-1
      DO 116 L=1,MK
      LDE(L)=LDES(L)
  116 CONTINUE
      DO 118 LJ=1,NDF
      DO 120 KJ=1,NDF
  120 RWA(KJ)=ROW(KJ,LJ)
      IF(LJ.LE.NDF-1.OR.NDF.EQ.1)MK1=K-1
      DO 122 JJ=LJ,MK1
      DO 124 L=LJ,MK2
      IF(LDEST(JJ).NE.LDES(L))GO TO 124
      LDES(L)=0
      DO 126 KJ=1,NDF
  126 ROW(KJ,JJ)=ROW(KJ,JJ)-RW(LJ,L)*RWA(KJ)
      GO TO 128
  124 CONTINUE
  128 CONTINUE
  122 CONTINUE
      IF(LJ.NE.1)GO TO 166
      K=JJ
      KA=LDEST(K)
      DO 130 KJ=1,NDF
      DO 130 IJ=1,NRHS
  130 RA(KJ,IJ)=ROW(KJ,K+IJ-1)
      DO 132 M=1,MK2
      IF(LDES(M).EQ.0)GO TO 132
      LDEST(K)=LDES(M)
      LDES(M)=0
      DO 134 KJ=1,NDF
  134 ROW(KJ,K)=-RW(LJ,M)*RWA(KJ)
      K=K+1
  132 CONTINUE
      LDEST(K)=KA
      DO 136 KJ=1,NDF
      DO 136 IJ=1,NRHS
  136 ROW(KJ,K+IJ-1)=RA(KJ,IJ)
  166 CONTINUE
      DO 138 KJ=1,NDF
```

```
      DO 138 IJ=1,NRHS
  138 ROW(KJ,K+IJ-1)=ROW(KJ,K+IJ-1)-RW(LJ,MK+IJ-1)*RWA(KJ)
      IF(LJ.EQ.NDF)GO TO 140
      DO 142 II=1,MK
      LDES(II)=LDE(II)
  142 CONTINUE
  140 CONTINUE
  118 CONTINUE
      KTEST=-1
      CALL SORT
      K=K-NDF
      GO TO 144
  146 IF(LL.EQ.1)K=MK1+1
      DO 148 LJ=1,NDF
      DO 148 KJ=1,NDF
      DO 148 IJ=1,NRHS
  148 ROW(KJ,K+IJ-1)=ROW(KJ,K+IJ-1)-ROW(KJ,LJ)*RW(LJ,IJ+1)
      KTEST=0
      CALL SORT
      K=K-NDF
  144 CONTINUE
      GO TO 112
  150 CONTINUE
      IF(LL.EQ.1)K=MK1+1
      DO 152 JK=1,NDF
      KN=K+NRHS-1
      DO 154 I=JK,KN
      RW(JK,I)=ROW(JK,I)/ROW(JK,JK)
  154 CONTINUE
      JL=JK+1
      IF(JL.GT.NDF)GO TO 156
      DO 158 JP=JL,NDF
      RWA(JP)=ROW(JP,JK)
      DO 160 I=JK,KN
      ROW(JP,I)=ROW(JP,I)-RW(JK,I)*RWA(JP)
  160 CONTINUE
  158 CONTINUE
  156 CONTINUE
  152 CONTINUE
      JJ=NEP-1
  162 IF(K1.EQ.JJ)GO TO 164
      READ(1)
      K1=K1+1
      GO TO 162
  164 K=K+NRHS-1
      WRITE(1)NDF,K,(LDEST(I),I=1,K),((RW(JH,I),I=1,K)
     1,JH=1,NDF)
      WRITE(6,710)NEP,K
C
      GO TO 99
  199 CONTINUE
C
      NODV=NUMVAR
C
C     BACKSUBSTITUTION
C
      WRITE(6,711)
      DO 200 IJ=1,NRHS
      WRITE(6,712)IJ
      WRITE(6,715)
```

```
      IF(IJ.EQ.1)GO TO 209
      NUMVAR=NODV
      REWIND 1
      DO 208 JJK =1,NUMNP
  208 READ(1)
  209 CONTINUE
      BACKSPACE 1
      READ(1)NDF,MK,(LDEST(I),I=1,MK),((RW(JH,I),I=1,MK)
     1,JH=1,NDF)
      BACKSPACE 1
      MK=MK-NRHS+1
      IF(IJ.EQ.1)GO TO 201
      DO 202 JJK=1,NDF
      RW(JJK,MK)=RW(JJK,MK+IJ-1)
  202 CONTINUE
  201 CONTINUE
      IF(LDEST(MK).NE.1)GO TO 212
      DO 210 I=1,NDF
      M=NDF+1-I
      A(NUMVAR)=RW(M,MK)
      NUMVAR=NUMVAR-1
  210 CONTINUE
      GO TO 220
  212 CONTINUE
      A(NUMVAR)=RW(NDF,MK)/RW(NDF,MK-1)
      NUMVAR=NUMVAR-1
      LK=MK-1
      LJ=NDF-1
      DO 222 JJ=1,LJ
      M=NDF-JJ+1
      NN=M-1
      AW=0.0
      DO 224 I=M,LK
      AW=AW+RW(NN,I)*A(LDEST(I))
  224 CONTINUE
      A(NUMVAR)=(RW(NN,MK)-AW)/RW(NN,NN)
      NUMVAR=NUMVAR-1
  222 CONTINUE
      NUMVAR=NUMVAR-1
  220 CONTINUE
      DO 230 J=2,NUMNP
      BACKSPACE 1
      READ(1)NDF,MK,(LDEST(I),I=1,MK),((RW(JH,I),I=1,MK)
     1,JH=1,NDF)
      BACKSPACE 1
      MK=MK-NRHS+1
      IF(IJ.EQ.1)GO TO 204
      DO 205 JJK=1,NDF
      RW(JJK,MK)=RW(JJK,MK+IJ-1)
  205 CONTINUE
  204 CONTINUE
      IF(LDEST(MK).NE.1)GO TO 232
      DO 234 I=1,NDF
      M=NDF-I+1
      A(NUMVAR)=RW(M,MK)
      NUMVAR=NUMVAR-1
  234 CONTINUE
      GO TO 230
  232 CONTINUE
      LK=MK-1
```

```
      DO 240 JJ=1,NDF
      M=NDF-JJ+2
      NN=M-1
      AW=0.0
      DO 242 I=M,LK
      AW=AW+RW(NN,I)*A(LDEST(I))
  242 CONTINUE
      A(NUMVAR)=(RW(NN,MK)-AW)/RW(NN,NN)
      WRITE(6,713)NUMVAR,A(NUMVAR)
      NUMVAR=NUMVAR-1
  240 CONTINUE
  230 CONTINUE
C
      WRITE(8,714)(A(I),I=1,NODV)
C     WRITE(6,714)(A(I),I=1,NODV)
  200 CONTINUE
C
  700 FORMAT(1X,'THE FOLLOWING NODES HAVE SPECIFIED BOUNDARY
     1CONDITIONS')
  701 FORMAT(10I8)
  702 FORMAT(2X,'ASSEMBLY/ELIMINATION BEGINS')
  703 FORMAT(10I8)
  704 FORMAT(3E12.5)
  705 FORMAT(2I8,3E12.5)
  706 FORMAT(10E12.4)
  707 FORMAT(10I8)
  708 FORMAT(I8)
  709 FORMAT(1X,'NODE NO. NONZERO ENTRIES IN ROW')
  710 FORMAT(2I8)
  711 FORMAT(2X,'BACKSUBSTITUTION BEGINS')
  712 FORMAT(2X,'BACKSUBSTITUTION STEP #',I6,'    BEGINS')
  713 FORMAT(I8,E12.5)
  714 FORMAT(3E12.5)
  715 FORMAT(1X,'VAR. NO.    RESULT')
      STOP
      END
C
      SUBROUTINE SORT
      COMMON/C1/LDEST(177),ROW(3,177),M,KTEST,NDF,MAXN,NRHS
      NP=0
      KK=NDF+1
      M=M+NRHS-1
      DO 10 I=KK,M
      NP=NP+1
      LDEST(NP)=LDEST(I)
      DO 12 L=1,NDF
   12 ROW(L,NP)=ROW(L,I)
   10 CONTINUE
      IF(KTEST.EQ.0)GO TO 26
      NRB=M-2*NDF-NRHS+1
      NRC=NRB-NDF
      DO 20 I=1,NRC,NDF
      KA=MAXN
      N=I+NDF
      DO 22 J=N,NRB,NDF
      IF(LDEST(I).LT.LDEST(J))GO TO 22
      IF(KA.LT.LDEST(J))GO TO 22
      KA=LDEST(J)
      K=J
   22 CONTINUE
```

```
      IF(KA.NE.MAXN)GO TO 24
      GO TO 20
   24 CONTINUE
      DO 40 KJ=1,NDF
      MM=I-1+KJ
      LL=K-1+KJ
      K1=LDEST(MM)
      LDEST(MM)=LDEST(LL)
      LDEST(LL)=K1
      DO 40 LJ=1,NDF
      R1=ROW(LJ,MM)
      ROW(LJ,MM)=ROW(LJ,LL)
      ROW(LJ,LL)=R1
   40 CONTINUE
   20 CONTINUE
   26 M=M-NRHS+1
      RETURN
      END
```

REFERENCES

1. B. M. Irons, 'A frontal solution program for finite element analysis', *Int. j. numer. methods eng*, **2**, 5–32 (1970).
2. P. Hood, 'Frontal solution program for unsymmetrical matrices', *Int. j. numer. methods eng*, **10**, 379–399 (1976).
3. E. Thompson and Y. Shimazaki, 'A frontal procedure using skyline storage', *Int. J. numer. methods eng*, **15**, 889–910 (1980).
4. S. K. Gupta and K. K. Tanji, 'Computer program for solution of large sparse, unsymmetric systems of linear equations', *Int. j. numer. methods eng*, **11**, 1251–1259 (1977).
5. A. Recuero and J. P. Gutierrez, 'A direct linear system solver with small core requirements', *Int. j. numer. methods eng*, **14**, 633–645 (1979).