KERNFORSCHUNGSANLAGE JÜLICH GmbH

Zentralabteilung Allgemeine Technologie

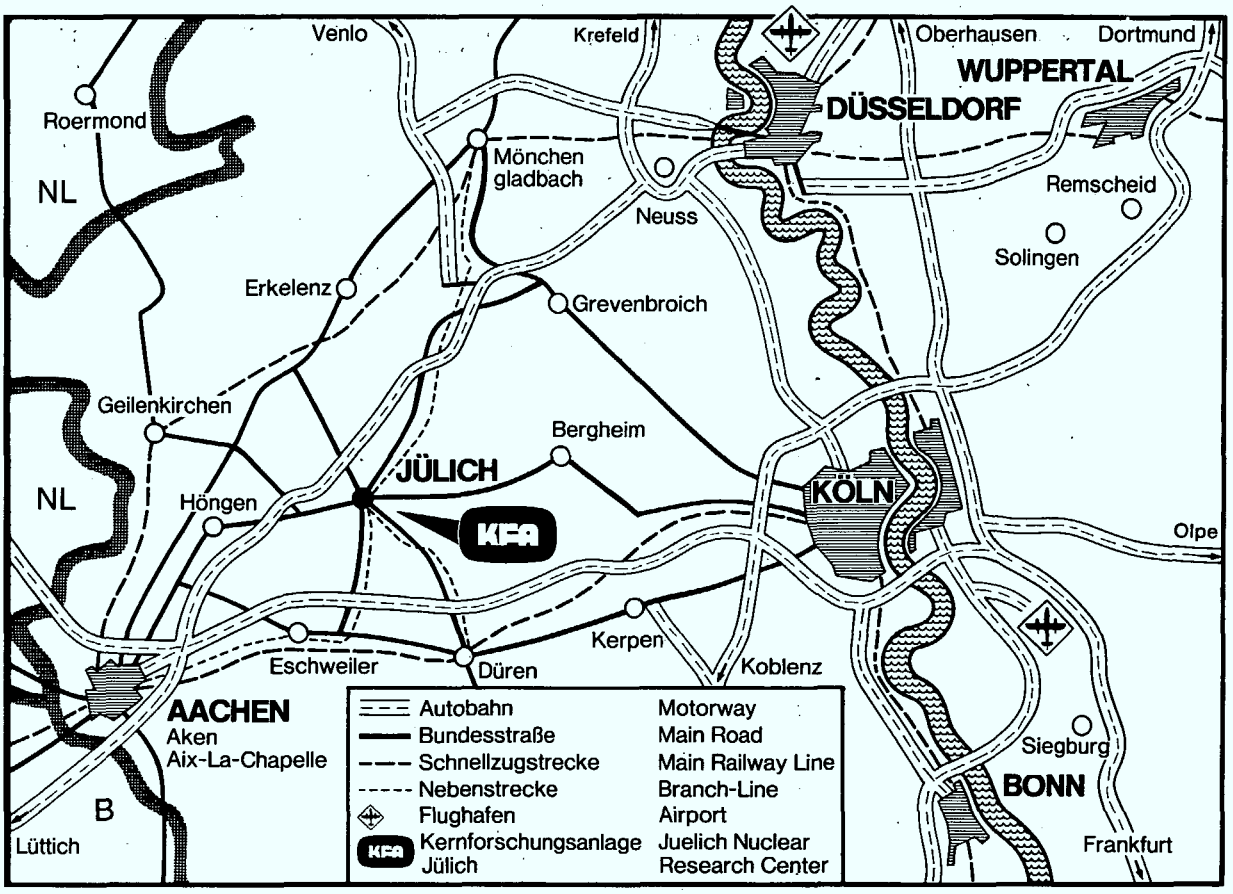
Experiences with a Matrix Bandwidth Reduction Method Applied to a Finite Element Problem

by

P. Kukkonen

Jül - 1326
Juli 1976

Als Manuskript gedruckt



Berichte der Kernforschungsanlage Jülich – Nr. 1326

Zentralabteilung Allgemeine Technologie Jül - 1326

Im Tausch zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH, Jülich, Bundesrepublik Deutschland

Experiences with a Matrix Bandwidth Reduction Method Applied to a Finite Element Problem

by

P. Kukkonen
Helsinki University of Technology[1])

[1]) Guest coworker in the Zentralabteilung Allgemeine Technologie der Kernforschungsanlage Jülich

Summary

Large sparsely populated matrices of diagonal character are common in finite element calculations. Computer time is saved and exactness improved if the matrix bandwidth is small. Therefore it is useful to apply a bandwidth reduction procedure to finite element programs, which renumbers the nodal points with the goal of minimum differences between neighboured node numbers.
A bandwidth reduction algorithm created by Collins was tested. Some improvements were implemented and its applicability extended to large elements, with 8 nodes instead of 4. A test run dealt with a real technical structural problem. The experiences exhibit an excellent efficiency. It is to recommend to append this procedure as prerunning one to all kinds of finite element programs.

1. Introduction

Finite element and finite differences calculations move along with the establishing of large sparsely populated symmetrical matrices of diagonal character, the so called stiffness matrices. These matrices must be inverted. The procedure increases largely in its time effort, and decreases in its exactness, the broader the non zero population is on both sides of the matrix diagonal, the matrix bandwidth. In finite element calculations the bandwidth only depends on the numbering sequence of the nodal points, and bandwidth reduction means nothing else but skillful nodal point renumbering. The bandwidth becomes the smaller, the less the differences are between neighboured node numbers.
R.J.Collins presents /1/ a method for bandwidth reduction by automatic renumbering. The two subprograms, which he describes have been used here in connection with a special problem. Our object was to try the usefulness of his algorithm in the case of 8-node finite elements, especially despite to the fact, that Collins doubts about a successful application concerning elements with more than 4 nodal points.

2. The Problem

The special problem was to reduce the bandwidth in a practical case. The case occured in the context with calculations of a torus section of the plasma containment of a fusion machine, see fig. 1. This problem is a part of a calculation parcel concerning the TEXTOR project, which is executed by the Finite Element Working Group in the subdivison for Engineering calculations of the ZAT. The structure was subdivided in 176 hexaedral finite elements with eight nodes each. There was a total number of 426 nodes.

3. The Program

It's here not our task to describe the program details in an explicit manner. This is done in R.S.Collins paper. On the

other hand, there is a certain demand for flow charts of the two important subprograms, which help for an easy understanding and are completed here. These subroutines are called SETUP and OPTNUM.A list and a run of the program can be seen in the appendix, as well as the flow charts.

3.1 The main program

The main program does nothing else but calls the subroutines. The variables needed in the bandwidth reduction algorithm are presented to the routines through a COMMON area. It is arbitarily dimensioned for 500 nodes.

3.2 The subprograms

3.2.1 RD

The relationships between elements and nodes are put in. This step has its importance, too, if Collins method shall be applied to other problems. The node numbers of each element must be placed in an array called JT so that the nodes of an element with the number n have the following indices:
n, 500+n, 1000+n, 1500+n.... JT is dimensioned to (500x8) because 8-node elements are handled.

3.2.2 SETUP

This subprogram fills two arrays, JMEM and MEMTJ for later use in OPTNUM. The number of related nodes to one special node is stored in JMEM. For example the memory place JMEM (n) contains, how many nodes are related to the node n. In MEMTJ are then the numbers of these related nodes. MEMTJ is arranged in such a way that the related nodes to node n have indices:
(n-1)x27+1, (n-1)x27+2, (n-1)x27+3,..., (n-1)x27+k, where k is equal to JMEM (n). JMEM is dimensioned to (500) and MEMTJ to (27x500). In this problem the maximum amount of related nodes is 27. An example how JMEM and MEMTJ look like follows. The configuration can be seen in fig. 2.

```
JMEM(1) = 3        MEMTJ(1-27)   = 2,5,6,-,-,-,-,...   (- = undefined)
JMEM(2) = 3        MEMTJ(28-54)  = 5,1,3,-,-,-,-,...
JMEM(3) = 3        MEMTJ(55-81)  = 5,2,4,-,-,-,-,...
JMEM(4) = 4        MEMTJ(82-108) = 5,3,7,8,-,-,-,...
  .                  .
  .                  .
  .                  .
```

3.2.3 OPTNUM

This is the most important part of the program. A node renumbering trying system takes place to gain a bandwidth reduction. The algorithm is following: the program starts once from every node and relabels it as numer one. After that the array MEMTJ is taken for help. The neighbouring nodes which are stored there are now renumbered in that order they are located in MEMTJ. Then the program jumps to the node with the new next higher number, now number two, and relabels its neighbours, which don't have a new number already, again with help of MEMTJ. After this the new node three is taken and so on. If the difference between two related nodes becomes larger than that of the best achieved configuration, including the original, the routine is stopped and started again by numbering the next node as number one. The new numbers are stored in the array JNT, which is dimensioned again to (500). The working principle of the method is tried to explain with an example below: For the configuration see fig. 3.

The contents of the arrays:

Considered node	JMEM	MEMTJ
1)	2	2,8
2)	4	3,5,8,1
3)	3	4,5,2
4)	2	3,5
5)	6	2,3,4,6,7,8
6)	2	5,7
7)	3	5,6,8
8)	4	1,2,5,7

Supposing that the program has already tried to start with nodes 1,2,3 it is now the turn to relabel 4 as 1, see fig.4.

The renumbering system runs according this scheme. Let us call the basic node, the neighbour relations of which are considered, the key node.
The old node 4 gets the new number (1), new number in brackets, and becomes the first key node, written on the left edge.

(1) had the old neighbours 3 & 5, the numbers of which now are renamed with the next free numbers of the new numbering system

3 → (2)
5 → (3).

(2) As the next key node serves the number, which follows in the sequence of numbers to that of the former key node. This is the new nodal number (2). The old neighbours were 4,5,2. 4(1) and 5(3) are already renumbered.The old 2 receives the next higher free number of the new system, which is (4)

2 → (4).

(3) Now we proceed to the node number (3) as new key node, the old numbers of its neighbours were 2(4), 3(2), 4(1), 6,7,8. To the old numbers 6,7 and 8 are assigned the succeeding free new numbers

6 → (5)
7 → (6)
8 → (7).

(4) The next step deals with the next key node (4); neighbours were 3(2), 5(3), 8(7), 1,

1 → (8).

etc.

Which node's neighbours are renumbered (the new number)	MEMTJ for this node (old numbers)	New number	Old number
1	3,5	2	3
		3	5
2	④,⑤,2	4	2
3	②,③,④,6,7,8	5	6
		6	7
		7	8
4	③,⑤,⑧,1	8	1

(○ means that the node has already been relabeled)

See also fig.4, where the above steps can be seen schematically.

3.2.4 MATRIX

In this subprogram new arrays are created for the new configuration with the same properties as JMEM and MEMTJ possess. After that the subroutine TAULU is called with data of both the new and the old numbering.

3.2.5 TAULU

Here the pattern of the matrix is built by using the arrays JMEM and MEMTJ. Because in this case the matrices are very large (426x426), they are not completely displayed. Instead of this a smaller matrix, which consists of (4x4) submatrices is built. But 426 divided by 4 does lead to an integer. Therefore we apply the following scheme. In the corners the submatrices are (6x6) and on the edges (6x4) or (4x6). In this way very much array area is saved and the matrix can be printed on two pages. The resulting matrix is (106x106).

3.2.6 TULOS

The results are printed here. First we receive a list of the old and new nodal numbers, and then schematic pictures of the old and the new matrix. Also the bandwidths are written.

3.2.7 A list of the variables in SETUP and OPTNUM

Name	Type	Dimension	Purpose	
NODES	1*2		number of the nodes	SO
LMENTS	1*2		number of the elements	SO
JT	1*2	4000	nodes of each element	SO
MEMTJ	1*2	13500	the related nodes for every node	SO
JMEM	1*2	500	the number of related nodes for every node	SO
JNT	1*2	500	the new node numbers, located equal to the old ones	SO
IDIFF	1*2		the original bandwidth	SO
I,II,III,J,JJ	1*2		indices	S
JNTI	1*2		identifies one node of JT	S
JSUB	1*2		pointer for the array MEMTJ	S
JJT	1*2		identifies one node of JT	S
MEM1	1*2		identifies one node of JMEM	S
I,K	1*4		indices	O
MEWJT	1*2	500	the old node numbers, located equal to the new ones, during an attempt	O
JOINT	1*2	500	the new node numbers, located equal to the old ones, during an attempt	O
MINIMAX	1*2		the bandwidth of the best achieved configuration	O
MAX	1*2		the biggest value of bandwidth during a renumbering attempt	O
K4	1*2		old number of that node, whose neighbours are relabeled	O
K5	1*2		old number of that node, to which is assigned a new number	O
NDIFF	1*2		the difference between two related nodes	O

S = subroutine SETUP
O = subroutine OPTNUM

4. Additional Remarks and Advices to the User

We found that there was the necessity of four improvements.

Subroutine SETUP

1) In the beginning of the routine IDIFF is initialized to NODES. It causes the program to believe that the original bandwidth is equal to NODES. Because of this the new configuration may be worse than the original one. The correct command must be IDIFF = 0 or IDIFF = 1.

2) The command DO 60 J = 1, LMENTS may lead to miscalculations. In Collins' example it worked, but it is possible to make a configuration where it doesn't do that, for example, if 4-node elements are used in a structure as in fig.5. In this case the program reads only the 3 first elements, because it tries to find an element, which has node number three located in the same way as node 1 in element 1 and node 2 in element 2. So node number 9 is not read, because the system supposes there to be only 4 elements and has added one further element by itself. If this command is written DO 60 J = 1, NODES, the routine perhaps does some unnecessary additional work, but at least it considers every element and every node. After this change the variable LMENTS becomes unnecessary and could be removed from the program.

Subroutine OPTNUM

3) The function subprogram IABS has to be defined separately to integer * 4, because the command IMPLICIT INTEGER * 2(I-N) otherwise makes it I*2 and not all compilers accept that.

4) There is a command IF (K.EQ.NJTS) 60 TO 50. A variable NJTS does not exist elsewhere in the program, but NODES has to occupy its place.

It is very easy to get an impression from (1) that when using 4-node elements, one node can have only 8 neighbours. This is not correct. For example in the configuration of fig.6 node

number 5 has 10 neighbours. In a symmetrical case with an 8-node element one node has 26 neighbours. But in our example the configuration is not symmetric, and so there are 4 nodes with 28 neighbours. Because of that, when the subprograms SETUP and OPTNUM are used with a new problem, it is advicable to print out the contents of the array JMEM in the first run. The simplest way to do this is to write the following commands in the end of SETUP:

```
      DO 80 I = 1, NODES
      WRITE (6,70) JMEM (I)
70    FORMAT (5X, I2)
80    CONTINUE
```

When seeing the list it is easy to pick up the biggest number and if it is bigger than the supposed symmetric value 26, change 26 to this number everywhere in the program. Also the dimensions of MEMTJ have to be changed to 500 x maximum value of related nodes.

The subprogram SETUP can also be used in connection with an other method, which Akhnas and Dhatt present (2). MEMTJ and JMEM are just the arrays that the authors claim when their minimizing routine begins.

The two subprograms SETUP and OPTNUM can of course also be handled as a "black box". It is not so important to understand how they work. Three things have to be remembered when using these routines:

1) It is necessary to input the values of the variables NODES and LMEMTJ.
2) Nodes of every element should be given in the array JT arranged as explained before.
3) The array JNT contains then the new node numbers, their locations in the array being equal to the old node numbers.

The subprograms SETUP and OPTNUM are listed separately behind the example program (see appendix) in such a mode, that they can be added to a problem, where bandwidth reduction is needed.

5. Results and Conclusions

As a look on the symbolic population matrices, see Fig.7 , shows, the bandwidth in the treated case was reduced from 364 to 57, obviously an excellent result.
This result is encouraging to recommend the procedure, consisting of SETUP and OPTNUM, as a prerunning additional program to every finite element analysis program.

6. Acknowledgements

The author wishes to express his gratitude to Dr.Stelzer for giving the theme and helpful advices, and A.Sievers for support in handling the Jülich IBM-computer.

7. References

(1) R.J.Collins: Bandwidth reduction by automatic renumbering, International Journal for Numerical Methods in Engineering 6, 345-356 (1973)

(2) G. Akhnas and G. Dhatt: An automatic node relabeling scheme for minimizing a matrix or network bandwidth, International Journal for Numercial Methods in Engineering 10, 787-797 (1976)

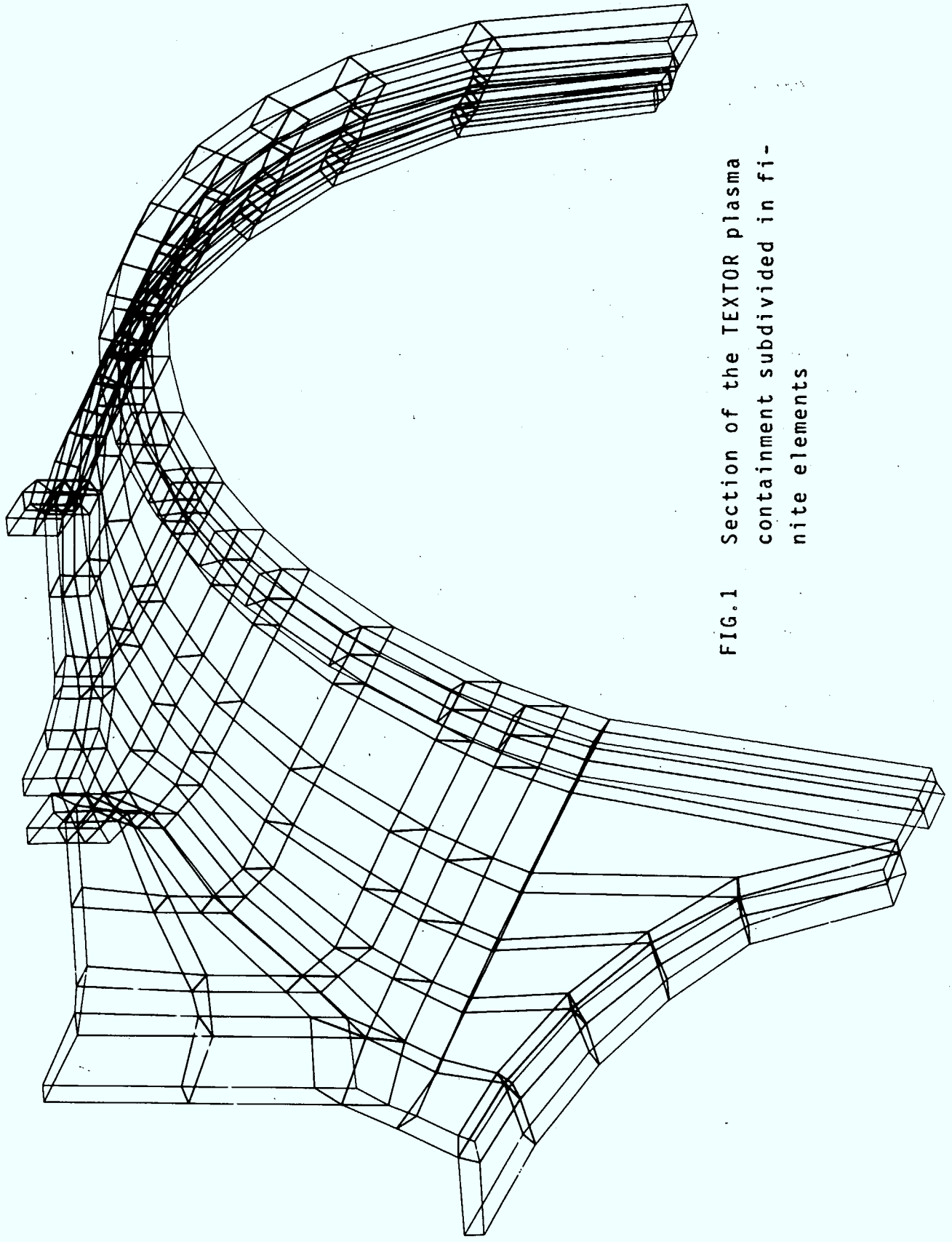


FIG.1 Section of the TEXTOR plasma containment subdivided in finite elements

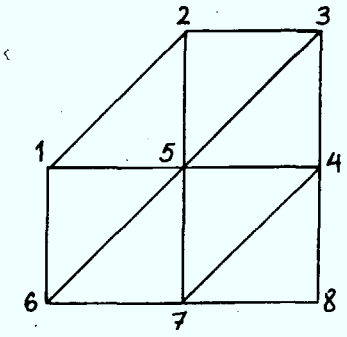


Fig.2 (3-node elements)

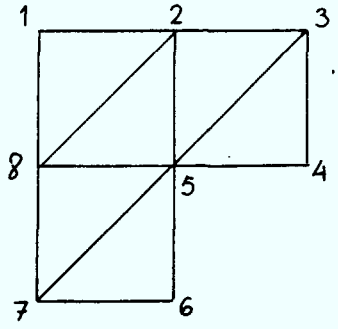


Fig.3 (3-node elements)

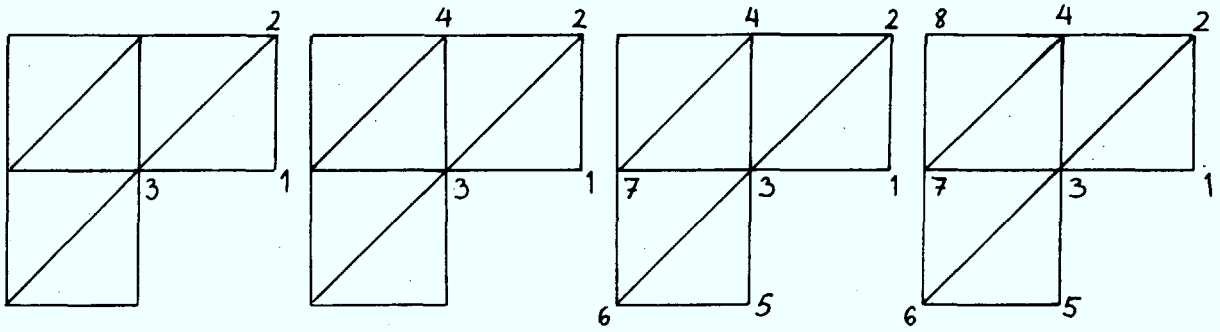


Fig.4 a, b, c, d

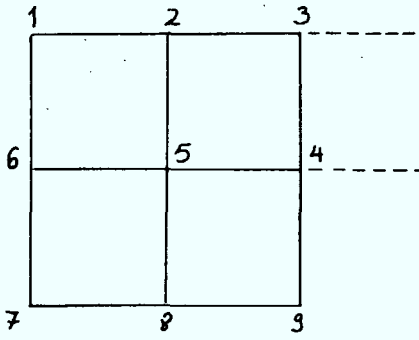


Fig.5

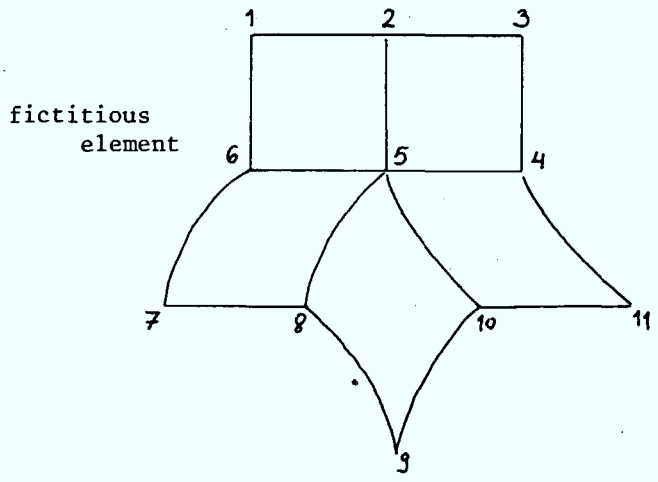


Fig.6

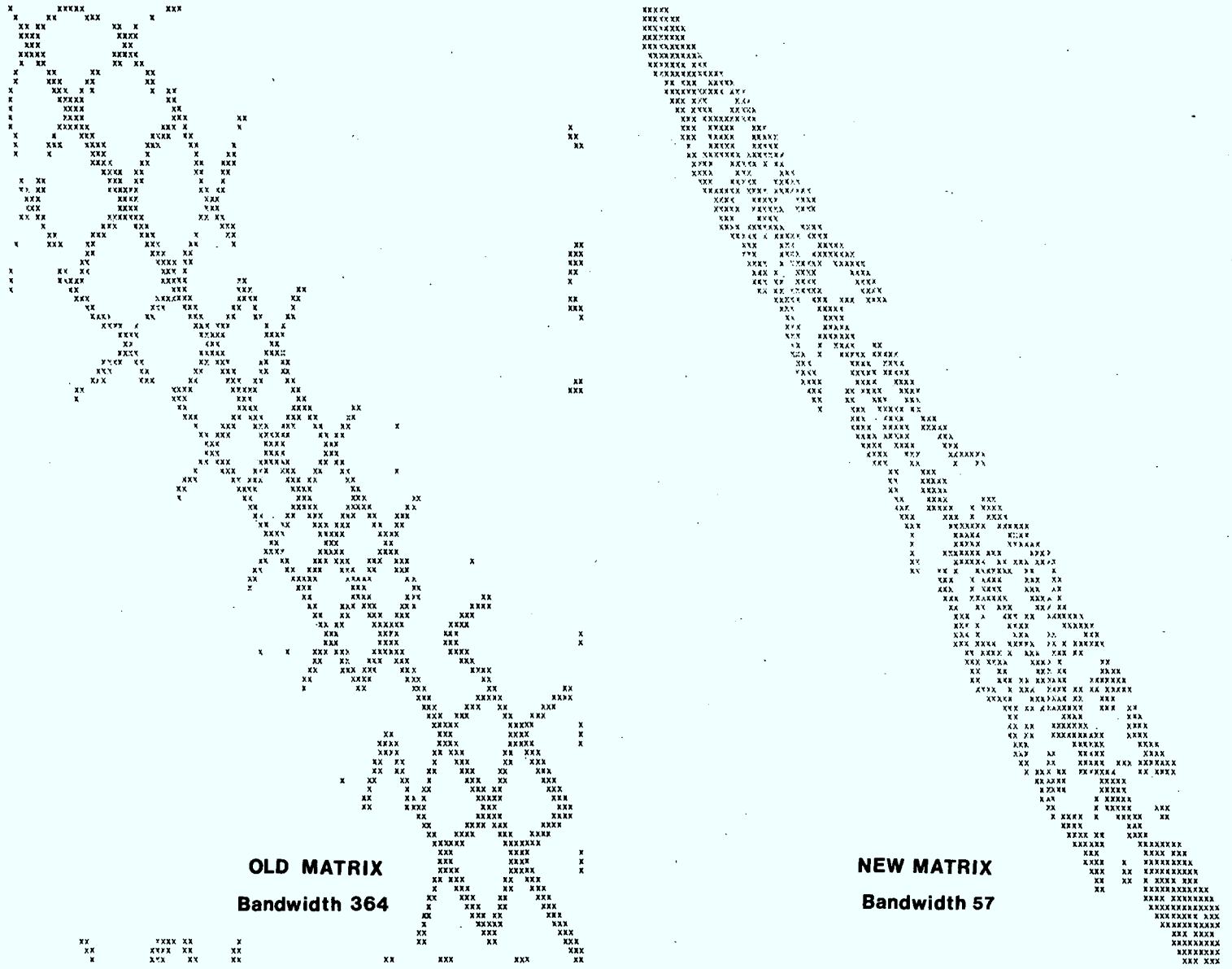
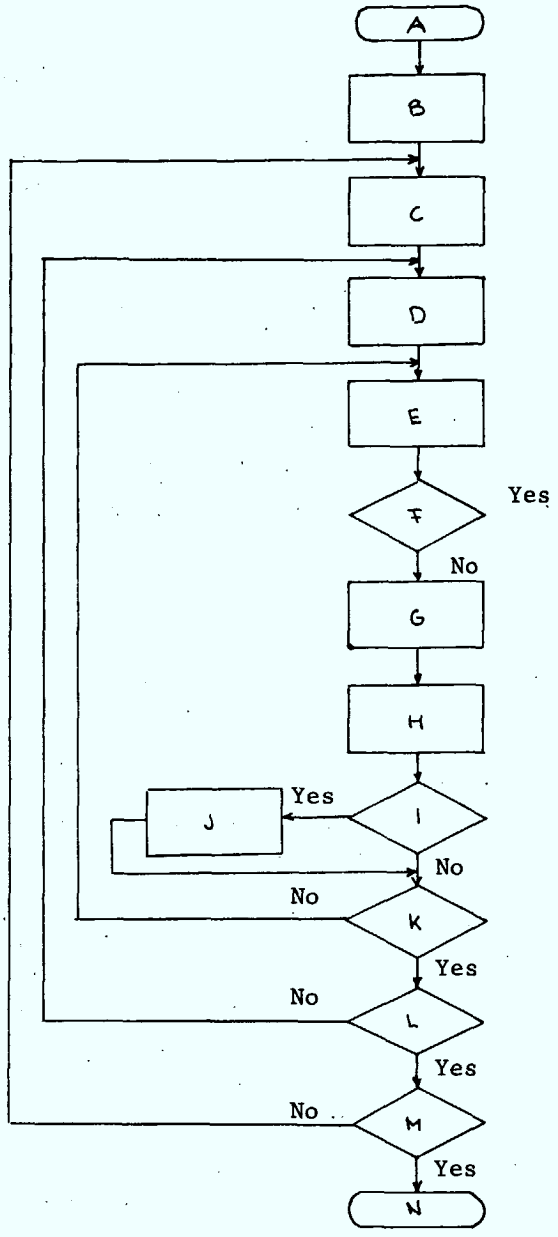


Fig.7 Comparison between the original and the improved matrix

*** THE BANDWIDTH IS 57 ***

SUBROUTINE SETUP



A begin

B the bandwidth and the array JMEM are initialized to zero

C next element is taken

D next node (1) of the element is initialized

E next node (2) that is in the same element as (1) is initialized

F has the relationship between (1) and (2) been established?

G 1 is added to the place in JMEM corresponding (1) and (2) is put in a place in MEMTJ corresponding (1)

H the difference (3) between two the related nodes (1)&(2) is calculated

I is (3) bigger than the biggest bandwidth calculated before?

J bandwidth = (3)

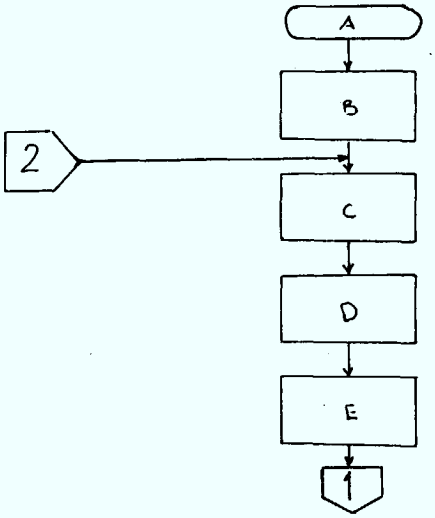
K has the whole element been considered?

L has the whole element been considered?

M have all the elements been considered?

N end

SUBROUTINE OPTNUM



A begin

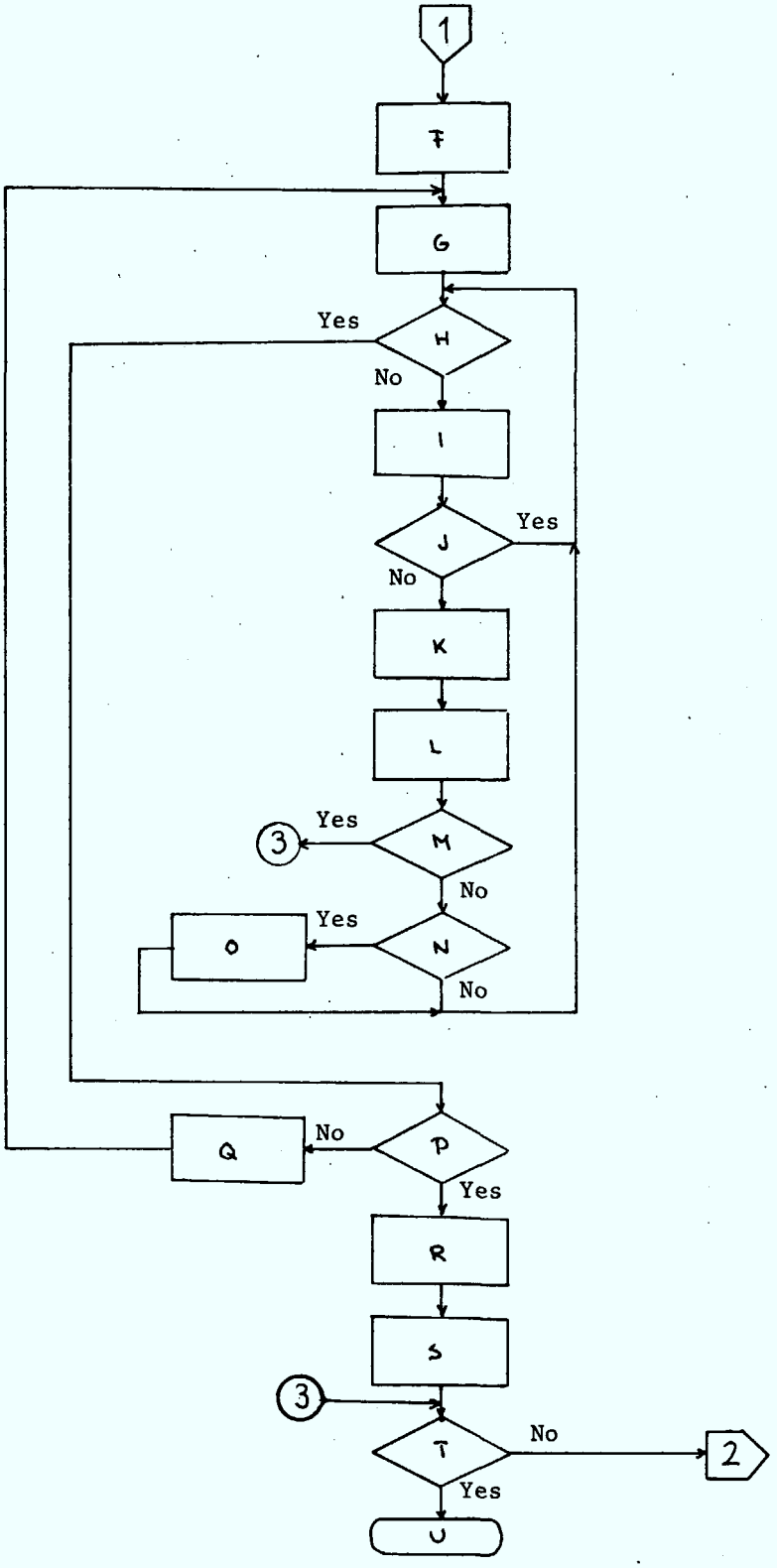
B the bandwidth of the best achieved configuration (1) is initialized to the original bandwidth

C the helping matrizes, with which the renumbering is done and the biggest bandwidth (2) of this attempt are initialized to zero

D counter I is initialized to I

E the next node is taken and given the new number one

SUBROUTINE OPTNUM, CONTINUATION



F counter K is initialized to one

G the number of related nodes to the new node I is taken

H have all related nodes been considered ?

I next related node is localised in the MEMTJ array

J does the node already have a new number?

K 'K=K+1, the new number of the node = K is put in the memory

L the difference (3) between J and K is counted

M is this difference (3) larger than the bandwidth (1) of the best before achieved configuration

N is this difference (3) larger than the bandwidth (2) of this renumbering attempt?

O the largest value of the bandwidth in this attempt (J)

P is K equal NODES, that is, are all nodes renumbered?

Q I = I+1

R the best achieved bandwidth (1) is given the value of (2)

S the new configuration is put in the array JNT

T has the program started from every node,relabeling it as number one?

U end

APPENDIX 1 THE PROGRAM USED IN THE CASE OF THE EXAMPLE

```
      IMPLICIT INTEGER*2(I-N)
      COMMON NODES,LMENTS,JT(4000),MEMTJ(13500),JMEM(500),JNT(500),IDIFF
      INTEGER*2 OLDMAT(106,106),FINMAT(106,106)
      NODES=426
      LMENTS=176
      CALL RD(JT)
      CALL SETUP
      CALL OPTNUM(LEVEYS)
      CALL MATRIX(OLDMAT,FINMAT)
      CALL TULOS(OLDMAT,FINMAT,LEVEYS)
      STOP
      END

      SUBROUTINE RD(JT2)
      IMPLICIT INTEGER*2(I-N)
      DIMENSION JT1(8,500),JT2(4000)
      DO 5 I=1,8
      DO 5 J=1,500
5     JT1(I,J)=0
      DO 10 J=1,176
      READ(5,50)(JT1(I,J),I=1,8)
50    FORMAT(1X,8(6X,I3))
10    CONTINUE
      DO 20 I=1,8
      DO 20 J=1,500
      JJ=500*(I-1)+J
20    JT2(JJ)=JT1(I,J)
      WRITE(6,68)
68    FORMAT(1H1,4X,'NUMBER OF ELEMENT',35X,'NODES IN THIS ELEMENT'///)
      DO 100 J=1,176
      WRITE(6,70) J,(JT1(I,J),I=1,8)
70    FORMAT(10X,I3,15X,8(7X,I3))
100   CONTINUE
      RETURN
      END

      SUBROUTINE SETUP
      IMPLICIT INTEGER*2(I-N)
      INTEGER*4 JNTI,JJT,J1,IABS
      COMMON NODES,LMENTS,JT(4000),MEMTJ(13500),JMEM(500),JNT(500),IDIFF
      IDIFF=0
      DO 10 J=1,NODES
10    JMEM(J)=0
      DO 60 J=1,NODES
      DO 50 I=1,8
      JNTI=JT(500*(I-1)+J)
      IF (JNTI.EQ.0) GO TO 60
      JSUB=(JNTI-1)*27
      DO 40 II=1,8
      IF (II.EQ.I) GO TO 40
      JJT=JT(500*(II-1)+J)
      IF (JJT.EQ.0) GO TO 50
      MEM1=JMEM(JNTI)
      IF (MEM1.EQ.0) GO TO 30
      DO 20 III=1,MEM1
      M=MEMTJ(JSUB+III)
      IF (M.EQ.JJT) GO TO 40
20    CONTINUE
30    JMEM(JNTI)=JMEM(JNTI)+1
      MM=JSUB+JMEM(JNTI)
      MEMTJ(MM)=JJT
      J1=IABS(JNTI-JJT)
      IF (J1.GT.IDIFF) IDIFF=J1
40    CONTINUE
```

```
      50  CONTINUE
      60  CONTINUE
          RETURN
          END

          SUBROUTINE OPTNUM(LEVEYS)
          IMPLICIT INTEGER*2(I-N)
          INTEGER*4 I,K,IABS
          COMMON NODES,LMENTS,JT(4000),MEMTJ(13500),JMEM(500),JNT(500),IDIFF
          DIMENSION NEWJT(500),JOINT(500)
          MINMAX=IDIFF
          DO 60 IK=1,NODES
          DO 20 J=1,NODES
          JOINT(J)=0
      20  NEWJT(J)=0
          MAX=0
          I=1
          NEWJT(1)=IK
          JOINT(IK)=1
          K=1
      30  K4=JMEM(NEWJT(I))
          IF (K4.EQ.0) GO TO 45
          JSUB=(NEWJT(I)-1)*27
          DO 40 JJ=1,K4
          K5=MEMTJ(JSUB+JJ)
          IF (JOINT(K5).GT.0) GO TO 40
          K=K+1
          NEWJT(K)=K5
          JOINT(K5)=K
          NDIFF=IABS(I-K)
          IF (NDIFF.GE.MINMAX) GO TO 60
          IF (NDIFF.GT.MAX) MAX=NDIFF
      40  CONTINUE
          IF (K.EQ.NODES) GO TO 50
      45  I=I+1
          GO TO 30
      50  MINMAX=MAX
          DO 55 J=1,NODES
      55  JNT(J)=JOINT(J)
      60  CONTINUE
          LEVEYS=MINMAX
          RETURN
          END

          SUBROUTINE MATRIX(OLDMAT,FINMAT)
          IMPLICIT INTEGER*2(I-N)
          COMMON NODES,LMENTS,JT(4000),MEMTJ(13500),JMEM(500),JNT(500),IDIFF
          INTEGER*2 OLDMAT(106,106),FINMAT(106,106),MEMTJ2(13500),NEWJT(500)
          INTEGER*2 JMEM2(500)
          DO 15 I=1,NODES
          LL=JNT(I)
      15  NEWJT(LL)=I
          DO 20 I=1,NODES
          III=NEWJT(I)
          IIII=JMEM(III)
          JMEM2(I)=IIII
          IF (IIII.EQ.0) GO TO 25
          DO 20 J=1,IIII
          KK1=(III-1)*27+J
          K2=MEMTJ(KK1)
          IF (K2.EQ.0) GO TO 20
          K3=JNT(K2)
          KK4=(I-1)*27+J
          MEMTJ2(KK4)=K3
      20  CONTINUE
```

```
126      25  CONTINUE
127          CALL TAULU(OLDMAT,MEMTJ,JMEM,NODES)
128          CALL TAULU(FINMAT,MEMTJ2,JMEM2,NODES)
129          RETURN
130          END

131          SUBROUTINE TAULU(MATR,MEM,JME,NODES)
132          IMPLICIT INTEGER*2(I-N)
133          DIMENSION MATR(106,106),MEM(13500),JME(500),M(4,426)
134          DO 10 I=1,106
135          DO 5 I1=1,4
136          DO 5 J1=1,426
137      5   M(I1,J1)=0
138          DO 20 II=1,4
139          III=(I-1)*4+II+1
140          IKL=JME(III)
141          DO 30 J=1,IKL
142          JJ=MEM((III-1)*27+J)
143          IF (JJ.EQ.0) GO TO 30
144          M(II,JJ)=1
145      30  CONTINUE
146      20  CONTINUE
147          DO 40 J=1,106
148          DO 50II=1,4
149          DO 50 JJ=1,4
150          JJJ=(J-1)*4+JJ+1
151          IKLU=M(II,JJJ)
152          IF (IKLU.EQ.1) GO TO 60
153      50  CONTINUE
154          MATR(I,J)=0
155          GO TO 40
156      60  MATR(I,J)=1
157      40  CONTINUE
158      10  CONTINUE
159          DO 15 J=1,426
160      15  M(1,J)=0
161          IKL=JME(1)
162          DO 70 J=1,IKL
163          JJ=MEM(J)
164          IF (JJ.EQ.0) GO TO 70
165          M(1,JJ)=1
166      70  CONTINUE
167          DO 80 J=1,106
168          DO 90 JJ=1,4
169          JJJ=(J-1)*4+JJ+1
170          IKLU=M(1,JJJ)
171          IF (IKLU.EQ.1) GO TO 100
172      90  CONTINUE
173          GO TO 80
174      100 MATR(1,J)=1
175          MATR(J,1)=1
176      80  CONTINUE
177          DO 16 J=1,426
178      16  M(1,J)=0
179          IKL=JME(426)
180          DO 110 J=1,IKL
181          JJ=MEM(425*27+J)
182          IF (JJ.EQ.0) GO TO 110
183          M(1,JJ)=1
184      110 CONTINUE
185          DO 120 J=1,106
186          DO 130 JJ=1,4
187          JJJ=(J-1)*4+JJ+1
188          IKLU=M(1,JJJ)
189          IF (IKLU.EQ.1) GO TO 140
```

```
190      130 CONTINUE
191          GO TO 120
192      140 MATR(106,J)=1
193          MATR(J,106)=1
194      120 CONTINUE
195          IF (M(1,1).EQ.0) GO TO 150
196          MATR(1,106)=1
197          MATR(106,1)=1
198      150 CONTINUE
199          DO 160 J=1,106
200      160 MATR(J,J)=1
201          RETURN
202          END

203          SUBROUTINE TULOS(OLD,NEW,LEVEYS)
204          IMPLICIT INTEGER*2(I-N)
205          INTEGER*2 OLD(106,106),NEW(106,106)
206          INTEGER*2 IAPU1(71,6),IAPU2(71,6),NEWJT(500)
207          COMMON NODES,LMENTS,JT(4000),MEMTJ(13500),JMEM(500),JNT(500),IDIFF
208          DO 10 I=1,NODES
209          LL=JNT(I)
210      10  NEWJT(LL)=I
211          WRITE(6,70)
212      70  FORMAT(1H1////36X,'***  OLD NODE NUMBERS - NEW NODE NUMBERS  ***'
            */////)
213          DO 20 I=1,71
214          DO 30 J=1,6
215          IJ=(I-1)*6+J
216          IAPU1(I,J)=JNT(IJ)
217      30  IAPU2(I,J)=IJ
218          WRITE(6,71)(IAPU2(I,J),IAPU1(I,J),J=1,6)
219      71  FORMAT(6(5X,I3,3H - ,I3,5X)//)
220      20  CONTINUE
221          LEV=IDIFF+1
222          WRITE(6,75) ((OLD(I,J),J=1,106),I=1,106)
223      75  FORMAT(1H1//40X,'***   THE OLD MATRIX   ***'///106('+',106I1//))
224          WRITE(6,82) LEV
225      82  FORMAT(/////37X,'***  THE BANDWIDTH IS ',I3,'  ***')
226          LEV=LEVEYS+1
227          WRITE(6,77) ((NEW(I,J),J=1,106),I=1,106)
228      77  FORMAT(1H1//40X,'***   THE NEW MATRIX   ***'///106('+',106I1//))
229          WRITE(6,84) LEV
230      84  FORMAT(////37X,'***  THE BANDWIDTH IS ',I3,'  ***'/1H1)
231          RETURN
232          END

     $ENTRY
```

APPENDIX "2": THE BANDWIDTH REDUCTION PROGRAM IN ITS GENERALLY USABLE SHAPE

```
0000100       IMPLICIT INTEGER*2(I-N)
0000500       IMPLICIT INTEGER*2(I-N)
0000600       COMMON NODES,LMENTS,JT(4000),MEMTJ(13000),JMEM(500),JNT(500),IDIFF
0000800       NODES=???
0000900       LMENTS=???
0001000       CALL RD(JT)
0001100       CALL SETUP
0001200       CALL OPTNUM
0001500       STOP
0001600       END
0001700       SUBROUTINE RD(JT2)
0001800       IMPLICIT INTEGER*2(I-N)
0001900       DIMENSION JT1(8,500),JT2(4000)
0002000       DO 5 I=1,8
0002100       DO 5 J=1,500
0002200     5 JT1(I,J)=0
0002300       DO 10 J=1,176
0002400       READ(5,50)(JT1(I,J),I=1,8)
0002500    50 FORMAT(1X,8(6X,I3))
0002600    10 CONTINUE
0002700       DO 20 I=1,8
0002800       DO 20 J=1,500
0002900       JJ=500*(I-1)+J
0003000    20 JT2(JJ)=JT1(I,J)
0003100       RETURN
0003200       END
0003300       SUBROUTINE SETUP
0003400       IMPLICIT INTEGER*2(I-N)
0003500       INTEGER*4 JNTI,JJT,J1,J2,IABS
0003600       COMMON NODES,LMENTS,JT(4000),MEMTJ(13000),JMEM(500),JNT(500),IDIFF
0003700       IDIFF=0
0003800       DO 10 J=1,NODES
0003900    10 JMEM(J)=0
0004000       DO 60 J=1,NODES
0004100       DO 50 I=1,8
0004200       JNTI=JT(500*(I-1)+J)
0004300       IF (JNTI.EQ.0) GO TO 60
0004400       JSUB=(JNTI-1)*20
0004500       DO 40 II=1,8
0004600       IF (II.EQ.I) GO TO 40
0004700       JJT=JT(500*(II-1)+J)
0004800       IF (JJT.EQ.0) GO TO 50
0004900       MEM1=JMEM(JNTI)
0005000       IF (MEM1.EQ.0) GO TO 30
0005100       DO 20 III=1,MEM1
0005200       M=MEMTJ(JSUB+III)
0005300       IF (M.EQ.JJT) GO TO 40
0005400    20 CONTINUE
0005500    30 JMEM(JNTI)=JMEM(JNTI)+1
0005600       MM=JSUB+JMEM(JNTI)
0005700       MEMTJ(MM)=JJT
0005800       J1=JNTI-JJT
0005900       J2=IABS(J1)
0006000       IF (J2.GT.IDIFF) IDIFF=J2
```

continued on the next page

```
C     40  CONTINUE
C     50  CONTINUE
C     60  CONTINUE
      RETURN
      END
      SUBROUTINE OPTNUM
      IMPLICIT INTEGER*2(I-N)
      INTEGER*4 I,K,IABS
      COMMON NODES,LMENTS,JT(4000),MEMTJ(13000),JMEM(500),JNT(500),IDIFF
      DIMENSION NEWJT(500),JOINT(500)
      MINMAX=IDIFF
      DO 60 IK=1,NODES
      DO 20 J=1,NODES
      JOINT(J)=0
   20 NEWJT(J)=0
      MAX=0
      I=1
      NEWJT(1)=IK
      JOINT(IK)=1
      K=1
   30 K4=JMEM(NEWJT(I))
      IF (K4.EQ.0) GO TO 45
      JSUB=(NEWJT(I)-1)*26
      DO 40 JJ=1,K4
      K5=MEMTJ(JSUB+JJ)
      IF (JOINT(K5).GT.0) GO TO 40
      K=K+1
      NEWJT(K)=K5
      JOINT(K5)=K
      NDIFF=IABS(I-K)
      IF (NDIFF.GE.MINMAX) GO TO 60
      IF (NDIFF.GT.MAX) MAX=NDIFF
   40 CONTINUE
      IF (K.EQ.NODES) GO TO 50
   45 I=I+1
      GO TO 30
   50 MINMAX=MAX
      DO 55 J=1,NODES
   55 JNT(J)=JOINT(J)
   60 CONTINUE
      RETURN
      END
```