REMARK: This user guide should be viewed and/or printed using a font such as Monaco or Courier (these fonts give the same width to all characters), size 12 pt., with 8 1/2 inch page width having 0.5 inch left and right margins.

FEMCOD - Program Description and User Guide

by

M. E. Plesha, R. D. Cook, and D. S. Malkus

Engineering Mechanics & Astronautics Program
Dept. of Nuclear Engineering & Engineering Physics
University of Wisconsin
Madison, WI 53706

March, 1988
revised 9.22.89

FEMCOD is a short but efficient finite element "program skeleton." It supplies the machinery and bookkeeping for input of data, assembly of elements, imposition of boundary conditions, solution of equations, and output of results. The user is expected to supply subroutines for element generation and post-processing (e.g. extraction of stresses from displacements). The program imposes no particular limitation on element type, shape, or number of nodes. Example elements are provided with the program.

FEMCOD is written in a style known as the "California school" of programming to which the first author was exposed by Professor W. K. Liu during his studies at Northwestern University. This style provides efficient programs that are neat and orderly, and are easy to read and modify. FEMCOD is essentially a simplified version of general-purpose production and research programs. Because it retains many of the features of these codes, it is an ideal program for first-time users and illustrates many useful programming techniques (in addition, it is efficient enough to be useful for many research applications).

Some of the features and limitations of the program are as follows:

(1) linear static analyses only.

(2) all elements of a particular structure must be the same type.

(3) capability for prescribed displacements and temperatures.

(4) the upper triangle of the symmetric stiffness matrix is stored using compact column (skyline) storage.

(5) an efficient "active column" equation solver is used.

(6) all REAL and INTEGER variables are stored sequentially in one-dimensional REAL and INTEGER arrays, respectively.*

(7) the storage capacity of the entire program is controlled by a single statement in the main program.

(8) the program is convenient for the analysis of 1-, 2-, and 3-dimensional problems in solid and structural mechanics, fluid mechanics and heat transfer.

---

*Larger codes usually store INTEGERs and REALs in the same one-dimensional array through the use of an EQUIVALENCE statement. However, some of the more "user-friendly" compilers, which students are likely to use, do not permit EQUIVALENCEing of INTEGERs and REALs so this feature is avoided in FEMCOD.

In what follows, we explain how the program works and how to use it. Use of the program does not require that all internal workings of the program be understood.

1. PROGRAM ARCHITECTURE

A FE program must perform a number of chores during the course of an analysis. Fortunately, each chore is, in itself, rather straightforward and easy to program. The computational procedure and subroutine names employed in FEMCOD are shown in Fig. 1.1.

		FEMCOD Subroutine name
(1)	Input control data	main
(2)	Initialize some of the memory pointers.	main
(3)	Check available memory.	MCHECK
(4)	Complete data input.	INPUT
(5)	Determine column heights for skyline storage of stiffness matrix.	COLHT
(6)	Initialize memory pointers for	

```
       stiffness matrix.                           main
 (7)  Check available memory.                      MCHECK
 (8)  Form global stiffness matrix:
       8.1  loop:  for each element
       8.2  form element stiffness matrix          user supplied
       8.3  add contribution to global
             stiffness matrix.                     ADSTIF and LOCSKY
       8.4  next element.
 (9)  Enforce prescribed displacement boundary     MODIFY
       conditions (enforcement of zero dis-
       placement boundary conditions is also
       partially done by ADSTIF in Step 8.3)
(10)  Triple-factor stiffness matrix.              TRFACT
(11)  Solve simultaneous equations by              TRFACT
       forward and back substitution.
(12)  Output solution.                             main
(13)  Post-process solution.                       user supplied
(14)  Stop.
```

Figure 1.1 Computational procedure and FEMCOD subroutine names.

2. COMPACTED 1-DIMENSIONAL ARRAY STORAGE

The program stores most REAL and INTEGER variables sequentially in one-dimensional REAL and INTEGER arrays that are dimensioned as "blank common" arrays A and IA. Memory pointers indicate the addresses in A and IA where individual blocks of data begin. For example, we wish to store nodal coordinate data and nodal external load data sequentially in a one-dimensional array, and we denote the memory pointers for this information by MPCORD and MPFEXT, respectively. If we assume that MPCORD starts at the first available word of memory then MPCORD = 1. Since there are NUMNP nodes and each has NSD spatial coordinates, nodal external load data can begin at MPFEXT = MPCORD + NUMNP*NSD. The memory maps for REAL and INTEGER array storage are shown in Fig. 2.1. A description of the information that is stored in the arrays of Fig. 2.1 is given in Appendix I. When inspecting the program listing for FEMCOD, note that the storage of the many blocks of data in arrays A and IA is only apparent in the main program: in subroutines, these blocks are dimensioned and used as separate arrays.

Notice in Fig. 2.1 that storage is allocated for NUMNP words of memory beginning at the location MPTEMP. This can be used for problems in which scalar-valued nodal point quantities are to be used. For example, in plane stress analysis, the nodal thicknesses can be stored there. In a thermal stress analysis, the nodal temperatures can be stored there. For other problems, this storage can be made larger if necessary, or if not needed, could be eliminated.

Also notice that in program FEMCOD, the subroutine DIMENSION statements have both row and column numbers. For example

```
DIMENSION  KFIX(NDOF, NUMNP)
```

appears frequently. It should be noted that Fortran requires only that the true row dimension be stated, thus,

```
DIMENSION KFIX(NDOF, 1)
```

would work equally as well, and indeed most finite element software is written this way. However, during program development and debugging, many compilers permit subscript checking which will not function unless both the row and column dimensions are accurately stated in the DIMENSION statement.

-----Changing the Amount of program Memory

All REAL and INTEGER variables are stored sequentially in the REAL and INTEGER arrays A and IA, respectively, which are dimensioned in a COMMON statement at the beginning of the main program. To change the program memory, simply change these dimensions. Also, don't forget to change MAXREL and MAXINT which are also at the beginning of the main program; these are used by subroutine MCHECK to assure that the program memory is not exceeded.

```
          MPCORD=1   MPFEXT     MPDISP     MPTEMP MPWORK     MPSTIF   MPEND
 REAL     -|---------|-----------|-----------|------|-----------|--------|-
 array A  -|---------|-----------|-----------|------|-----------|--------|-
            NUMNP*NSD NUMNP*NDOF  NUMNP*NDOF NUMNP  NUMNP*NDOF LENGTH

           IPKFIX=1   IPNOD       IPIADR     IPMAT   IPEND
 INTEGER   -|----------|-----------|-----------|-------|-------
 array IA  -|----------|-----------|-----------|-------|-------
             NUMNP*NDOF NUMEL*NNPE NUMNP*NDOF  NUMEL
```

Figure 2.1 One-dimensional array memory maps.

3. STIFFNESS MATRIX STORAGE SCHEME

FEMCOD assumes that the stiffness matrix is symmetric and stores the upper-triangular coefficients using a compact column, or skyline, storage scheme. For example, the coefficients of the stiffness matrix [K] are stored in the one-dimensional array [S] in the order

$$\begin{bmatrix} S(1) & S(2) & & S(5) & \\ & S(3) & & S(6) & \\ & & S(4) & S(7) & S(9) \\ & & & S(8) & S(10) \\ & & & & S(11) \end{bmatrix} \qquad (3.1)$$

The addresses in the one-dimensional array of the diagonal coefficients of [K] are contained in an array IADRES which, for Eq. 3.1 is given by

$$\text{IADRES} = [1, 3, 4, 8, 11] \qquad (3.2)$$

Thus, the address in S of a diagonal coefficient of matrix [K] is given by IADRES(K). Furthermore, the address of the uppermost coefficient in column k is given by IADRES(K-1) + 1 FOR 2 Û k Û n. The global equation number corresponding to the uppermost coefficient of column k is given by KEQTOP = K + IADRES(K-1) - IADRES(K) + 1. Also of interest is that the entire length of the one-dimensional array for storage of the stiffness matrix is LENGTH = IADRES(NEQ) where NEQ is the number of equations.

Although FEMCOD does not require it, a useful measure of the sparsity of an equation system is the mean, or average, semibandwidth M (called MBAND in FEMCOD). The mean semibandwidth is also useful for estimating the number of arithmetic operations during equation system factorization according to the expression nb(b/2), which was derived for large systems of equations stored in constant bandwidth form. The number of terms needed for constant semibandwidth storage of the upper (or lower) triangular part of the coefficient matrix is

$$L = nb - b(b-1)/2 \qquad (3.3)$$

where n is the number of equations and b is the (constant) bandwidth. To estimate the mean bandwidth M of skyline storage, we replace L by LENGTH and b by M in Eq. 3.3, and solve for M. This is done in subroutine COHLT, which computes and prints M. Hence, one can estimate the number of operations in factorization as nM(M/2).

To aid in assembling the global stiffness matrix from element stiffness matrices, a function subroutine LOCSKY is used. This function subroutine, given an I,J address within the skyline, returns the appropriate one-dimensional storage location. Furthermore, if the I,J address is outside of the skyline or below the diagonal, LOCSKY returns an address of 0.

4. IMPLEMENTATION OF USER ELEMENTS

FEMCOD places no restriction on the number of nodes a user-defined element may have, but requires that each node not have more than 6 degrees of freedom (NDOF Û 6). Generally, the user supplies a subroutine (or subroutines) which contain a DO LOOP within it over the number of elements.

The task of assembling the element stiffness matrix and the element load vector, if any, into the global stiffness matrix and global load vector is performed by subroutine ADSTIF. The calling instructions for this subroutine are fully explained in the subroutine's listing, but a few remarks are appropriate. Once the element stiffness matrix, SL, and the element load vector, if any, RE, for element number KELE is formed, subroutine ADSTIF is called as follows:

```
        CALL ADSTIF(KFIX,S,FEXT,NOD,IADRES,SL,RE,ISLROW,KELE,IOP)
```

where KFIX, NOD, and IADRES are explained in Appendix I, S is the global stiffness matrix, FEXT is the global load vector, and ISLROW is the row dimension of SL in the calling subroutine. If only an element stiffness matrix is to be assembled, then IOP should be set to zero (RE is not used in that case) and if both an element stiffness matrix and load vector are to be assembled, then IOP should be set to unity.

An example illustrating the implementation of an Euler-Bernoulli beam finite element is shown in Fig. 4.1.

```
      PROGRAM FEMCOD
          .
          .
C---- FORM STIFFNESS MATRIX
      CALL BEMSTF(A(MPCORD),IA(IPKFIX),IA(IPNOD),IA(IPIADR),IA(IPMAT),
     .            A(MPSTIF),A(MPFEXT))
          .
          .
          .
      SUBROUTINE BEMSTF(X,KFIX,NOD,IADRES,MATNUM,S,FEXT)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z),INTEGER(I-N)
      COMMON /KONTRL/ NUMNP,NDOF,NUMEL,NNPE,NSD,NEQ,LENGTH
      COMMON /MATRL / NMAT,RMAT(5,4)
      COMMON /DEVICE/ IIN,IOUT,IBUG
      DIMENSION X(NSD,NUMNP),KFIX(NEQ),NOD(NNPE,NUMEL),IADRES(NEQ),
     .          MATNUM(NUMEL),S(LENGTH),FEXT(NEQ),SL(6,6),T(6,6),
     .          RJUNK(6,6)
      DATA T/36*0./
C
C---- FORM STIFFNESS MATRIX FOR 2-DIMENSIONAL FRAME ELEMENTS (EULER-
C     BERNOULLI BEAM WITH AXIAL STIFFNESS)
C     RMAT(1,MAT)=C.S. AREA
C         (2,MAT)=MODULUS
C         (3,MAT)=INERTIA
C
      DO 100 N=1,NUMEL
      DO 5 I=1,6
      DO 5 J=I,6
    5 SL(I,J)=0.
      MAT=MATNUM(N)
      NODE1=NOD(1,N)
      NODE2=NOD(2,N)
C
C---- COMPUTE LENGTH, SINE AND COSINE OF BEAM ORIENTATION
      X21=X(1,NODE2)-X(1,NODE1)
      Y21=X(2,NODE2)-X(2,NODE1)
      RLONG=SQRT(X21*X21+Y21*Y21)
      COSINE=X21/RLONG
      SINE  =Y21/RLONG
C
C---- FORM ELEMENT STIFFNESS MATRIX IN LOCAL SYSTEM, SL
      TEMP1=RMAT(1,MAT)*RMAT(2,MAT)/RLONG
      TEMP2=RMAT(2,MAT)*RMAT(3,MAT)/(RLONG**3)
C
      SL(1,1)=TEMP1
      SL(1,4)=-SL(1,1)
      SL(2,2)=12.*TEMP2
      SL(2,3)=6.*RLONG*TEMP2
      SL(2,5)=-SL(2,2)
      SL(2,6)= SL(2,3)
      SL(3,3)=4.*RLONG*RLONG*TEMP2
      SL(3,5)=-SL(2,3)
      SL(3,6)=2.*RLONG*RLONG*TEMP2
      SL(4,4)=TEMP1
      SL(5,5)=SL(2,2)
      SL(5,6)=-SL(2,3)
      SL(6,6)= SL(3,3)
C
      DO 10 I=1,6
      DO 10 J=I,6
   10 SL(J,I)=SL(I,J)
C
```

```
C---- PERFORM TRANSFORMATION T(TRANSPOSE)*SL*T
      T(1,1)=COSINE
      T(1,2)=SINE
      T(2,1)=-SINE
      T(2,2)=COSINE
      T(3,3)=1.
      DO 20 I=1,3
      DO 20 J=1,3
   20 T(I+3,J+3)=T(I,J)
C
C---- MATRIX MULTIPLICATION RJUNK=SL*T
      DO 40 I=1,6
      DO 40 J=1,6
      SUM=0.
      DO 30 K=1,6
   30 SUM=SUM+SL(I,K)*T(K,J)
   40 RJUNK(I,J)=SUM
C
C---- MATRIX MULTIPLICATION SL=T(TRANSPOSE)*RJUNK
      DO 60 I=1,6
      DO 60 J=I,6
      SUM=0.
      DO 50 K=1,6
   50 SUM=SUM+T(K,I)*RJUNK(K,J)
      SL(I,J)=SUM
   60 SL(J,I)=SUM
C
C---- ASSEMBLE STIFFNESS MATRIX
      CALL ADSTIF(KFIX,S,FEXT,NOD,IADRES,SL,RE,6,N,0)
C
  100 CONTINUE
      RETURN
      END
```

Figure 4.1 Example of implementation of a subroutine in FEMCOD for the generation of a beam finite element stiffness matrix.

5. ENFORCEMENT OF DISPLACEMENT BOUNDARY CONDITIONS

To demonstrate how prescribed displacement boundary conditions are enforced, consider the following partitioned system of equations, in which each $\underset{\sim}{k}$, $\underset{\sim}{d}$, and $\underset{\sim}{f}$ represents a submatrix:

$$\begin{bmatrix} \mathbf{k}_{11} & \mathbf{k}_{12} & \mathbf{k}_{13} & \mathbf{k}_{14} & \mathbf{k}_{15} \\ \mathbf{k}_{21} & \mathbf{k}_{22} & \mathbf{k}_{23} & \mathbf{k}_{24} & \mathbf{k}_{25} \\ \mathbf{k}_{31} & \mathbf{k}_{32} & \mathbf{k}_{33} & \mathbf{k}_{34} & \mathbf{k}_{35} \\ \mathbf{k}_{41} & \mathbf{k}_{42} & \mathbf{k}_{43} & \mathbf{k}_{44} & \mathbf{k}_{45} \\ \mathbf{k}_{51} & \mathbf{k}_{52} & \mathbf{k}_{53} & \mathbf{k}_{54} & \mathbf{k}_{55} \end{bmatrix} \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \\ \mathbf{d}_4 \\ \mathbf{d}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \mathbf{f}_4 \\ \mathbf{f}_5 \end{bmatrix} \tag{5.1}$$

In Eq. 5.1 we will consider $\{\mathbf{d}_2\}$ and $\{\mathbf{d}_4\}$ to be prescribed displacements, and and $\{\mathbf{d}_1\}$, $\{\mathbf{d}_3\}$ and $\{\mathbf{d}_5\}$ are to be determined. Noting that $\{\mathbf{d}_2\}$ and $\{\mathbf{d}_4\}$ are known, the first row of Eq. 5.1 can be expanded to give

$$[\mathbf{k}_{11}]\{\mathbf{d}_1\} + [\mathbf{k}_{13}]\{\mathbf{d}_3\} + [\mathbf{k}_{15}]\{\mathbf{d}_5\} = \{\mathbf{f}_1\} - [\mathbf{k}_{12}]\{\mathbf{d}_2\} - [\mathbf{k}_{14}]\{\mathbf{d}_4\} \tag{5.2}$$

with similar expressions for the third and fifth rows of Eq. 5.1. The second and third rows of Eq. 5.1 are then written as

$$[\mathbf{I}]\{\mathbf{d}_2\} = \{\mathbf{d}_2\} \quad \text{and} \quad [\mathbf{I}]\{\mathbf{d}_4\} = \{\mathbf{d}_4\} \tag{5.3}$$

The final system of equations then becomes

$$\begin{bmatrix} \mathbf{k}_{11} & \mathbf{0} & \mathbf{k}_{13} & \mathbf{0} & \mathbf{k}_{15} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{k}_{31} & \mathbf{0} & \mathbf{k}_{33} & \mathbf{0} & \mathbf{k}_{35} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{k}_{51} & \mathbf{0} & \mathbf{k}_{53} & \mathbf{0} & \mathbf{k}_{55} \end{bmatrix} \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \\ \mathbf{d}_4 \\ \mathbf{d}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 - \mathbf{k}_{12}\mathbf{d}_2 - \mathbf{k}_{14}\mathbf{d}_4 \\ \mathbf{d}_2 \\ \mathbf{f}_3 - \mathbf{k}_{32}\mathbf{d}_2 - \mathbf{k}_{34}\mathbf{d}_4 \\ \mathbf{d}_4 \\ \mathbf{f}_5 - \mathbf{k}_{52}\mathbf{d}_2 - \mathbf{k}_{54}\mathbf{d}_4 \end{bmatrix} \tag{5.4}$$

There are several possible methods for implementing Eq. 5.4 with a compact column storage scheme. In program FEMCOD, the column heights associated with d.o.f. that are prescribed to be zero (i.e., zero-displacement boundary conditions), is one. Thus only the diagonal coefficient is stored. Enforcement of zero-displacement boundary conditions takes place in subroutine COLHT, in which the column height is set to one if the d.o.f. has prescribed zero value, in subroutine ADSTIF, in which the assembly of element stiffness coefficients corresponding to zero displacement d.o.f. is skipped, and in subroutine MODIFY, in which a unit number is placed on the diagonal of the stiffness matrix for d.o.f. with prescribed zero value.

In program FEMCOD, the column heights associated with nonzero prescribed displacements are the same as if the d.o.f. were not prescribed. After assembly of the global stiffness matrix, the load vector is modified for nonzero prescribed displacements as per Eq. 5.4. Then the coefficients in the columns above the diagonal of d.o.f. that are prescribed are set to zero. FEMCOD could be made slightly more efficient by doing one of the following:

(1) After modifying the load vector for nonzero prescribed displacements, the column heights of prescribed displacement d.o.f. could be reduced to one. This would avoid trivial multiplications involving zero stiffness coefficients during the solution of simultaneous equations.

(2) Column heights associated with zero and nonzero prescribed displacements could be set to one at the outset (i.e., in subroutine COLHT). To account for nonzero prescribed displacements, the load vector would be modified on the element-level. In other words, during the formation of element stiffness matrices, the load vector is modified as per Eq. 5.4. This requires more arithmetic operations to enforce the nonzero prescribed displacements, but wastes no storage for the compacted storage of the global stiffness matrix.

6. SOLUTION OF EQUILIBRIUM EQUATIONS

The system of simultaneous algebraic equations

$$[A]\{x\} = \{b\} \tag{6.1}$$

is solved using triple factoring. In Eq. 6.1, [A] is the stiffness matrix, {b} is the specified "load" vector, and {x} is the "displacement" vector to be determined. [A] is n by n, and {x} and {b} are n by 1 where n is the number of equations. In triple factoring, the matrix [A], which must be symmetric, is factored into the product of three matrices

^{T}

$$[A] = [U] [D][U] \tag{6.2}$$

where [D] is diagonal, and [U] is upper triangular with unit diagonal coefficients.

The algorithm is derived as follows. Consider the product $[U]^T [D][U] = [A]$ which can be written as

$$\begin{bmatrix} 1 & & & & \\ u_{12} & 1 & & & \\ u_{13} & u_{23} & 1 & & \\ u_{14} & u_{24} & u_{34} & 1 & \\ . & . & . & . & \\ . & . & . & . & \\ . & . & . & . & \end{bmatrix} \begin{bmatrix} d_{11} & & & & \\ & d_{22} & & & \\ & & d_{33} & & \\ & & & d_{44} & \\ & & & & . \\ & & & & . \\ & & & & . \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} & \ldots \\ & 1 & u_{23} & u_{24} & \ldots \\ & & 1 & u_{34} & \ldots \\ & & & 1 & \ldots \\ & & & & \\ & & & & \end{bmatrix} = [A] \tag{6.3}$$

Performing the multiplications in Eq. 6.3 provides

$$\begin{bmatrix} d_{11} & d_{11}u_{12} & d_{11}u_{13} & d_{11}u_{14} & \ldots \\ & (u_{12}d_{11}u_{12} + d_{22}) & (u_{12}d_{11}u_{13} + d_{22}u_{23}) & (u_{12}d_{11}u_{14} + d_{22}u_{24}) & \ldots \\ & & (u_{13}d_{11}u_{13} + u_{23}d_{22}u_{23} + d_{33}) & (u_{13}d_{11}u_{14} + u_{23}d_{22}u_{24} + d_{33}u_{34}) & \ldots \\ \text{symmetric} & & & & \\ & & & (u_{14}d_{11}u_{14} + & \ldots \end{bmatrix} = [A]$$

$$\left.\begin{matrix} u_{24}\, d_{22}\, u_{24} + \\ u_{34}\, d_{33}\, u_{34} + \\ d_{44}) \end{matrix}\right] \qquad (6.4)$$

An algorithm that computes the coefficients of [D] and [U] is shown in Fig. 6.1a with a version in which [D] and [U] overwrite [A] shown in Fig. 6.1b. Note that this algorithm computes coefficients of [U] in columnwise order. That is, for column k, the algorithm computes u_{1k}, u_{2k}, ..., $u_{k-1\,k}$, followed by d_{kk}.

(a)

```
$d_{11} = a_{11}$

- for  k = 2, ..., n
    - for  i=1, ..., k-1
        - $u_{ik} = \left[ a_{ik} - S_{m=1}^{i-1} u_{mi}\, d_{mm}\, u_{mk} \right] \frac{1}{d_{ii}}$
    - $d_{kk} = a_{kk} - S_{m=1}^{k-1} u_{mk}\, d_{mm}\, u_{mk}$
```

(b)

```
- for  k = 2, ..., n
    - for  i=1, ..., k-1
        - $a_{ik} = \left[ a_{ik} - S_{m=1}^{i-1} a_{mi}\, a_{mm}\, a_{mk} \right] \frac{1}{a_{ii}}$
    - $a_{kk} = a_{kk} - S_{m=1}^{k-1} a_{mk}\, a_{mm}\, a_{mk}$
```

Figure 6.1 (a) Algorithm for triple factoring of a symmetric matrix [A], (b) version in which [D] and [U] overwrite coefficients of [A]. Summations are denoted by S and are defined to be zero if the lower index is greater than the upper index.

The algorithm shown in Fig. 6.1b can be made more efficient for numerical implementation by noting that the product $a_{mm}\, a_{mk}$ appears inside two summations. An algorithm that takes this multiplication outside of the summations is shown in Fig. 6.2.

```
for $k=2, \ldots, n$
    for $i=1, \ldots, k-1$
        $a_{ik} = \left[ a_{ik} - \sum_{m=1}^{i-1} a_{mi} g_m \right] \frac{1}{a_{ii}}$
    $g_i = a_{ii} a_{ik}$
$a_{kk} = a_{kk} - \sum_{m=1}^{k-1} a_{mk} g_m$
```

Figure 6.2 More efficient version for numerical implementation of the algorithm shown in Figure 6.1b

The algorithm shown in Fig. 6.2 can be easily implemented using compacted column (skyline) storage for [A]. Since the summations that appear in Fig. 6.2 loop over coefficients of [A] in column-wise order, the ideal compacted storage scheme will store coefficients in column-wise order. The storage scheme described in Section 3 is ideally suited to the algorithm of Fig. 6.2.

Notice in Fig. 6.1 that division by d_{kk} is required. If the coefficient matrix [A] is singular or almost singular, then at some point in the factorization, division by zero, or a very small number will be attempted. To help detect this possibility, a test on $|d_{kk}/a_{kk}| < t$ is performed where t is a tolerance, a_{kk} is the original diagonal coefficient and d_{kk} is the pivot element. Taking $t = 10^{-p}$ will cause the factorization subroutine (TRFACT) to issue a warning if the leading p digits of any pivot element, d_{kk}, are lost compared to its original value, a_{kk}.

To determine whether [A] is positive definite, consider the scalar s given by

$$s = \{p\}^T [A]\{p\} \tag{6.5}$$

By definition, [A] is positive definite if $s > 0$ for any nontrivial vector {p}. Combining Eqs. (6.2) and (6.5) and defining {q} = [U]{p}, provides

$$s = \{q\}^T [D]\{q\} \tag{6.6}$$

Since [D] is diagonal, if all of its coefficients are positive, then $s > 0$, and hence [A] is positive definite. If any of the diagonal coefficients of [D] are zero or negative, then [A] is not positive definite. Although triple factoring will work for indefinite but symmetric nonsingular matrices [A], a test on positive definiteness is made in subroutine TRFACT.

Once factorization of [A] is complete, the solution {x} to the simultaneous equations is found as follows. Combining Eqs. 6.1 and 6.2 gives

$$[U]^T \{y\} = \{b\} \tag{6.7}$$

where

$$\{y\} = [D][U]\{x\} \tag{6.8}$$

Using forward substitution, Eq. 6.7 is solved for {y}. Then Eq. 6.8 is solved for {x} by first premultiplying by $[D]^{-1}$, which is trivial to compute since [D] is diagonal, and then back-substituting.

An algorithm for forward/back substitution is shown in Fig. 6.3a with a version in which the intermediate solution {y} overwrites {b} shown in Fig. 6.3b.

```
$y_1 = b_1$

for k = 2, ..., n
    $y_k = b_k - \sum_{m=k+1}^{k-1} u_{km} y_m$

$x_n = y_n / d_{nn}$

for k = n - 1, ..., 1
    $x_k = \frac{y_k}{d_{kk}} - \sum_{m=k+1}^{n} u_{km} x_m$
```

```
for k = 2, ..., n
    $b_k = b_k - \sum_{m=1}^{k-1} a_{mk} b_m$

$b_n = b_n / a_{nn}$

for k = n - 1, ..., 1
    $b_k = \frac{b_k}{a_{kk}} - \sum_{m=k+1}^{n} a_{km} b_m$
```

(a) (b)

Figure 6.3 (a) Algorithm for forward/back substitution, (b) version in which the intermediate solution {y} overwrites {b}, and [D] and [U] are stored in [A].

Note that in Fig. 6.3b, the summation over a_{mk} in the forward substitution phase uses the coefficients a_{mk} in columnwise order which is ideally suited to the compact column storage scheme used for [A]. However, the back substitution phase uses coefficients a_{km} in rowwise order which is not compatible with the storage scheme for [A]. To make back substitution more rapid and easy to program, modification as shown in Fig. 6.4 is made.

```
- for k = 2, ..., n
|
- $b_k = b_k - \sum_{m=1}^{k-1} a_{mk} b_m$

- for k = 1, ..., n
|
- $b_k = b_k / a_{kk}$

-   for k = n, ..., 2
|
|  - for m = 1, ..., k-1
|  |
---   $b_m = b_m - a_{mk} b_k$
```

Figure 6.4 Algorithm for forward/back substitution that exploits compacted storage.

APPENDIX I - Description of Major Variables

IIN	-	input file # (set to 5)
IOUT	-	output file # (set to 6)
MAXREL	-	dimension of real array A for sequential variable storage
MAXINT	-	dimension of integer array IA for sequential variable storage.
IBUG	-	debugging parameter for use in controlling output from subroutines WI and WR. When modifying program FEMCOD (e.g., adding a new element) it is often useful for the programmer to liberally add calls to subroutines WI and WR. For example, say the quantity AREA is computed and the programmer desires that its value be output. The following call to WR could be used: `CALL WR("AREA    ", AREA, 3)` In subroutine WR, IBUG is compared with 3. Then if IBUG is 3 or greater, AREA will be printed. To suppress the printing of AREA, the user can reset IBUG to a number less than 3. Generally speaking, IBUG=1 will give minimal output while higher values of IBUG will give more extensive output.
MPCORD	-	real array memory pointer for nodal coordinate data
MPFEXT	-	real array memory pointer for nodal forces and/or prescribed displacement.
MPDISP	-	real array memory pointer for nodal displacement solution.
MPTEMP	-	real array memory pointer for miscellaneous nodal-point quantities (e.g., temperatures).
MPWORK	-	real array memory pointer for work array.
MPSTIF	-	real array memory pointer for stiffness matrix.
MPEND	-	number of real storage words required by program (should be Û MAXREL).
IPKFIX	-	integer array memory pointer for node fixity data.

IPNOD	-	integer array memory pointer for element connectivity data.
IPIADR	-	integer array memory pointer for diagonal addresses of compact column storage of stiffness matrix.
IPMAT	-	integer array memory pointer for element material numbers.
IPEND	-	number of integer storage words required by program (should be Û MAXINT).
NUMNP	-	number of nodes.
NDOF	-	number of d.o.f./node.
NUMEL	-	number of elements.
NNPE	-	number of nodes per element.
NSD	-	number of space dimensions (1, 2 or 3). NSD is the number of space dimensions necessary to define a node point's position in space. For example, for a straight beam structure, NSD could be 1 or 2, depending on how the element generating subroutine operates. For a flat plate, NSD will usually be 2.
NMAT	-	number of material types.
NEQ	-	number of equations.
MBAND	-	mean stiffness matrix semi-bandwidth.
LENGTH	-	number of words of storage for stiffness matrix.
X	-	real array containing nodal point coordinates; dimension (NSD, NUMNP).
IADRES	-	integer array containing addresses of diagonal stiffness coefficients in compact column storage.
KFIX	-	integer array containing nodal fixity data; dimension (NSD, NUMNP). For example, for a nodal d.o.f. number M; 1 Û M Û NDOF, and node number N; 1 Û N Û NUMNP, KFIX(M,N) = 0 if d.o.f. is unconstrained = 1 if d.o.f. has prescribed zero value = 2 if d.o.f. has prescribed nonzero value
NOD	-	integer array containing element connectivity data;

		dimension (NNPE, NUMEL).
MATNUM	-	integer array containing element material numbers; dimension (NUMEL).
MAT	-	same as MATNUM.
S	-	one-dimensional real array containing global stiffness matrix; dimension (LENGTH).
F,FEXT	-	one-dimensional real arrays containing external nodal loads; dimension (NEQ).

APPENDIX II - Data Input

Line #1, title card (A80)

CARD

line #2, control card (6I5)

NUMNP, NDOF, NUMEL, NNPE, NSD, NMAT

remarks: NDOF Û 6
NSD Û 3
NMAT Û 4

next NUMNP pairs of lines, nodal data (I5, 4X, 6I1, 3F10.1, /, 15X, 6F10.0)

```
N, K1, K2, K3, K4, K5, K6, X, Y, Z
                           F1, F2, F3, F4, F5, F6
```

remarks: N is the node number and should be between 1 and NUMNP. A pair of lines must be supplied for each node, although they do not have to be in sequential order. Fixity codes (K1,K2, ...,K6) are defined as follows:

= 0 free
= 1 homogeneously constrained
= 2 nonzero prescribed

The ordering of the fixity codes should match the ordering of the element's d.o.f. in the element stiffness matrix generating subroutine.

Load/Prescribed-DOF values (F1,F2,..., F6) represent nodal loads (or nodal fluxes, etc.) when the corresponding fixity code is 0, and prescribed displacements (or nodal temperatures, etc.) when the corresponding fixity code is 2. The Load/Prescribed-DOF value must be zero if the corresponding fixity code is 1. For example, if K2=0, then F2 is the force applied to node N's second d.o.f. If K2=2, then F2 is the value of the prescribed displacement (or rotation) for node N's second d.o.f. If K2=1, then node N's second d.o.f. has a prescribed value of zero and F2 must be zero in the input statement.

Note: Although zero-prescribed d.o.f. values can be treated by using a fixity code of 2 and a zero value for the corresponding Load/ Prescribed-DOF number, the program is more efficient if a fixity code of 1 is used.

next NUMEL lines, element data (16I5)

N, MATNUM, NOD(1), NOD(2), ..., NOD(NNPE)

remark: include additional card for each node in 16I5 format if NNPE is greater than 14. N is the element number and should be between 1 and NUMEL. Element data should be supplied for each element, although it does not have to be supplied in sequential order.

next NMAT lines, material data (I5, 5F10.0)

MATNUM, RMAT(1,MATNUM), RMAT(2,MATNUM), .., RMAT(5,MATNUM)

remarks: RMAT(I,MATNUM) is selected by the user to represent material properties such as the MODULUS, POISSON'S RATIO, CONDUCTIVITY, ...

After FEMCOD reads the material data input, FE analysis begins automatically.

```
--------------------------------
APPENDIX III -  Sample Analysis
--------------------------------
```

Shown in Fig. A3.1 is a model of a beam consisting of two Euler-Bernoulli beam finite elements. The beam is built-in at the left-hand end, supports a midspan load of 350 lbs, and at the right-hand end, has a gap of 0.12 in. between the beam tip and a rigid support. Material and cross-section properties for the beam are also given in Fig. A3.1.

```
                                             cross section area = 0.12 in**2
     y                                       elastic modulus = 5.E04 psi
     |                      | P=350 lbs      moment of inertia = 1. in**4
     |                      |
     |                      |
    /|                      V
    /|---------------------------------------------
    /|                      |                     |
    /*  1                   *  2                  *  3    ----- x
    /|                      |                     |
    /|---------------------------------------------
    /|                                   0.12 in { _____
    /|                                            |\\\\
                                                  |\      *'s denote nodes
                                                  |\

    x=0. in                x=5. in                x=10. in
```

Figure A3.1 Two-element model of a cantilever beam with midspan load and a gap-support at the right-hand end.

In the analysis that follows, we will assume that the midspan load is sufficient to close the gap at the right-hand end of the beam and hence, we will prescribe a downward displacement of 0.12 in. for node 3 (the validity of this assumption should be checked by the user after the finite element analysis is completed by computing the reaction force at the right-hand end to verify that it is compressive, or by repeating the analysis with the beam tip unconstrained to verify that its tip displacement exceeds 0.12 in.).

The element-generating subroutine shown in Fig. 4.1 was used in FEMCOD for the analysis that follows. The data set that is read by FEMCOD is shown in Fig. A3.2. Note that the data line "123456789 12...." shown in Fig. A3.2 is not read by FEMCOD, but is simply a convenience to the user to aid in placing data in the correct fields for formatted reading.

```
EULER-BERNOULLI BEAM SAMPLE PROBLEM
    3    3    2    2    2    1
    1    111000        0.        0.        0.
                       0.        0.        0.
    2    000000        5.        0.        0.
                       0.     -350.        0.
    3    020000       10.        0.        0.
                       0.     -0.12        0.
    1    1    1    2
    2    1    2    3
    1      0.12     5.E04        1.

123456789 123456789 123456789 123456789 123456789 123456789 123456789
```

Figure A3.2 Data set read by FEMCOD for analysis of the model shown in Fig. A3.1.

The output produced by FEMCOD is shown in Fig. A3.3.

```
 EULER-BERNOULLI BEAM SAMPLE PROBLEM
```

```
PROGRAM CONTROL DATA
--------------------
NUMNP   =         3
NDOF    =         3
NUMEL   =         2
NNPE    =         2
NSD     =         2
NMAT    =         1
NEQ     =         9


REAL AND INTEGER ARRAY MEMORY POINTERS
-----------------------------------
MPCORD  =         1
MPFEXT  =         7
MPDISP  =        16
MPTEMP  =        25
MPWORK  =        28
MPSTIF  =        37
IPKFIX  =         1
IPNOD   =        10
IPIADR  =        14
IPMAT   =        23
IPEND   =        25
MAXREL  =     10000
MAXINT  =      1000


NODAL DATA  (#,X,Y,Z,K1 THRU K6,F1,F2,...)
----------
   1  .000E+00  .000E+00  .000E+00    111000  .000E+00  .000E+00  .000E+00
   2  .500E+01  .000E+00  .000E+00    000000  .000E+00 -.350E+03  .000E+00
   3  .100E+02  .000E+00  .000E+00    020000  .000E+00 -.120E+00  .000E+00


ELEMENT DATA  (#,MAT#,NODE1,NODE2,...)
------------
   1    1    1    2
   2    1    2    3


MATERIAL DATA  (#,RMAT1,RMAT2,...)
-------------
```

```
   1   .1200E+00   .5000E+05   .1000E+01   .0000E+00   .0000E+00


STIFFNESS MATRIX STORAGE PARAMETERS
-----------------------------------
MBAND   =         3
LENGTH  =        24
MPEND   =        61



NODAL POINT SOLUTION  (NODE #, NODAL DOF #, VALUE OF THE DOF)
--------------------
   1    1   .000000E+00      1    2   .000000E+00      1    3   .000000E+00
   2    1   .000000E+00      2    2  -.101302E+00      2    3  -.189688E-01
   3    1   .000000E+00      3    2  -.120000E+00      3    3   .387500E-02
```

Figure A3.3 Output listing produced by FEMCOD.

--the end--