CHAPTER ONE

Finite Element Method
The Big Picture

Computer Implementation 1.1 *(Matlab)*

The function for generating plane truss element equations can be implemented in *Matlab* in essentially the same way as that in *Mathematica*. Here is the *PlaneTrussElement* function in Matlab. The Matlab syntax requires that each function be saved in a separate text file with the name of the file same as that function name except with a ".m" extension.

MatlabFiles\Chap1\PlaneTrussElement.m

```
function k=PlaneTrussElement(e,A,coord)
% PlaneTrussElement(e,A,coord)
% Generates stiffness matrix for a plane truss element
% e=modulus of elasticity
% A=area of cross-section
% coord=coordinates at the element ends

x1=coord(1,1);y1=coord(1,2);
x2=coord(2,1);y2=coord(2,2);
L=sqrt((x2-x1)^2+(y2-y1)^2);
ls=(x2-x1)/L;ms=(y2-y1)/L;
k=e*A/L*[ls^2,ls*ms,-ls^2,-ls*ms;
        ls*ms,ms^2,-ls*ms,-ms^2;
        -ls^2,-ls*ms,ls^2,ls*ms;
        -ls*ms,-ms^2,ls*ms,ms^2];
```

For element number 6 (say with $E = 29 \times 10^6$ lb / in^2 and $A = 2.5$ in^2) of the transmission tower the equations are obtained as follows.

MatlabFiles\Chap1\TrussElementEx1.m

```
nodes=12*[-7.5,0;7.5,0;-7.5,15;
        7.5,15;-5,25;5,25;-5,35;5,35;-15,40;15,40;
        -25,45;-15,45;-5,45;5,45;15,45;25,45];
PlaneTrussElement(29000000,2.5,nodes([3 5],:))
```

```
>> TrussElementEx1

ans =

  1.0e+005 *

    0.3448    1.3791   -0.3448   -1.3791
    1.3791    5.5165   -1.3791   -5.5165
   -0.3448   -1.3791    0.3448    1.3791
   -1.3791   -5.5165    1.3791    5.5165
```

Computer Implementation 1.2 *(Matlab)*

The element equations for a triangular heat flow element can be generated conveniently by writing three simple functions. The following *HeatTriElement* function takes thermal conductivities (kx and ky), heat generation (Q) and the element nodal coordinates and returns the $\boldsymbol{k}_k$ matrix and the $\boldsymbol{r}_Q$ vector. A second function called *ConvectionTerm* generates $\boldsymbol{k}_h$ matrix and $\boldsymbol{r}_h$ vector and is needed when there is convection term specified. Three different versions of this function are defined depending on the side on which the convection boundary condition is specified. The third function called *HeatFluxTerm* computes $\boldsymbol{r}_q$ vector when a flux is specified along an element side.

MatlabFiles\Chap1\HeatTriElement.m

```
function[kk,rQ]=HeatTriElement(kx,ky,Q,coord)
%[kk,rQ]=HeatTriElement(kx,ky,Q,coord)
% Generates coefficient matrix of a triangular element for heat flow
% kx,ky=thermal conductivities in the x and y directions
% Q=Heat generation
% coord=coordinates at the element ends

x1=coord(1,1);y1=coord(1,2);
x2=coord(2,1);y2=coord(2,2);
x3=coord(3,1);y3=coord(3,2);
b1=y2-y3;b2=y3-y1;b3=y1-y2;
c1=x3-x2;c2=x1-x3;c3=x2-x1;
f1=x2*y3-x3*y2;f2=x3*y1-x1*y3;f3=x1*y2-x2*y1;
A=(f1+f2+f3)/2;
kxx=1/(4*A)*kx*[b1^2,b1*b2,b1*b3;
        b1*b2,b2^2,b2*b3;b1*b3,b2*b3,b3^2];
kyy=1/(4*A)*ky*[c1^2,c1*c2,c1*c3;
        c1*c2,c2^2,c2*c3;c1*c3,c2*c3,c3^2];
kk=kxx+kyy;
rQ=1/3*Q*A*[1;1;1];
```

MatlabFiles\Chap1\ConvectionTerm.m

```
function [kh, rh] = ConvectionTerm(side, h, Tinf, coord)
% [kh, rh] = ConvectionTerm(side, h, Tinf, coord)
% Generates kh and rh when convection is specified for a triangular element
% side = side over which the convection is specified
% h = convection coefficient
% Tinf = surrounding temperature
% coord = coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
switch (side)
case 1
    L=sqrt((x2-x1)^2+(y2-y1)^2);
    kh=h*L/6 * [2,1,0; 1,2,0; 0,0,0];
    rh=h*Tinf *L/2 * [1; 1; 0];
case 2
    L=sqrt((x2-x3)^2+(y2-y3)^2);
    kh=h*L/6 * [0,0,0; 0,2,1; 0,1,2];
    rh=h*Tinf *L/2 * [0; 1; 1];
case 3
    L=sqrt((x3-x1)^2+(y3-y1)^2);
    kh=h*L/6 * [2,0,1; 0,0,0; 1,0,2];
    rh=h*Tinf *L/2 * [1; 0; 1];
end
```

MatlabFiles\Chap1\HeatFluxTerm.m

```
function rq = HeatFluxTerm(side, q, coord)
% rq = HeatFluxTerm(side, q, coord)
% Generates vector resulting from a specified flux for a triangular element
% side = side over which the flux is specified
% q = flux value
% coord = coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
switch (side)
case 1
    rq = q*sqrt((x2 - x1)^2 + (y2 - y1)^2)/2 * [1; 1; 0];
case 2
    rq = q*sqrt((x2 - x3)^2 + (y2 - y3)^2)/2 * [0; 1; 1];
case 3
    rq = q*sqrt((x3 - x1)^2 + (y3 - y1)^2)/2 * [1; 0; 1];
end
```

Using these functions finite element equations for any triangular element for heat flow problem can easily be written. As an example we use these functions to develop matrices for the element number 20 in the finite element model of the heat flow through the L-shaped solid. The nodal coordinates for the entire model are are first defined. The element is connected between nodes 4, 10 and 5. With $k_x = k_y = 45$ and $Q = 5000000$, the $\boldsymbol{k}_k$ matrix and the $\boldsymbol{r}_Q$ vector for the element are generated by using the HeatTriElement function. There is an applied heat flux on side 3 of the element. With $q = 8000$, the $\boldsymbol{r}_q$ vector for the element is generated using the HeatFluxTerm function. The side 2 of the element is subjected to heat loss by convection. With $h = 55$ and $T_\infty = 20$, the $\boldsymbol{k}_h$ matrix and the $\boldsymbol{r}_h$ vector for the element are are generated by using the ConvectionTerm function. Adding matrices $\boldsymbol{k}_k$ and $\boldsymbol{k}_h$ and vectors $\boldsymbol{r}_Q$, $\boldsymbol{r}_q$, and $\boldsymbol{r}_h$ the complete element equations are as follows.

MatlabFiles\Chap1\HeatElementEx1.m

```
% Generation of element equations for a 2D heat flow problem
nodes = [0., 0.; 0., 0.0075; 0., 0.015; 0., 0.0225;
    0., 0.03; 0.015, 0.; 0.015, 0.0075; 0.015, 0.015;
    0.015, 0.0225; 0.015, 0.03; 0.03, 0.; 0.03, 0.0075;
    0.03, 0.015; 0.03, 0.0225; 0.03, 0.03; 0.045, 0.; 0.045, 0.0075;
    0.045, 0.015; 0.06, 0.; 0.06, 0.0075; 0.06, 0.015];
[kk, rQ] = HeatTriElement(45, 45, 5000000,nodes([4 10 5],:))
rq = HeatFluxTerm(3, 8000, nodes([4 10 5],:))
[kh, rh] = ConvectionTerm(2, 55, 20, nodes([4 10 5],:))
k = kk + kh
r = rq + rQ + rh
```

```
>> HeatElementEx1

kk =

   45.0000         0  -45.0000
         0   11.2500  -11.2500
  -45.0000  -11.2500   56.2500


rQ =

   93.7500
   93.7500
   93.7500


rq =

    30
     0
    30


kh =

         0         0         0
         0    0.2750    0.1375
         0    0.1375    0.2750


rh =

         0
    8.2500
    8.2500


k =

   45.0000         0  -45.0000
         0   11.5250  -11.1125
  -45.0000  -11.1125   56.5250
```

```
r =

  123.7500
  102.0000
  132.0000
```

Computer Implementation 1.3 *(Matlab)*

MatlabFiles\Chap1\PlaneStressTriElement.m

```
function k = PlaneStressTriElement(e, nu, h, coord)
% k = PlaneStressTriElement(e, nu, h, coord)
% Generates stiffness matrix for a triangular element for plane stress
% e = Modulus of elasticity
% nu = Poisson's ratio
% h = Thickness
% coord = Coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
b1 = y2 - y3; b2 = y3 - y1; b3 = y1 - y2;
c1 = x3 - x2; c2 = x1 - x3; c3 = x2 - x1;
f1 = x2*y3 - x3*y2; f2 = x3*y1 - x1*y3; f3 = x1*y2 - x2*y1;
A = (f1 + f2 + f3)/2;
C = e/(1 - nu^2)*[1, nu, 0; nu, 1, 0; 0, 0, (1 - nu)/2];
B = [b1, 0, c1; 0, c1, b1; b2, 0, c2; 0, c2, b2;
    b3, 0, c3; 0, c3, b3]/(2*A);
k = h*A*(B*C*B');
```

MatlabFiles\Chap1\PlaneStressTriLoad.m

```
function rq = PlaneStressTriLoad(side, qn, qt, h, coord)
% rq = PlaneStressTriLoad(side, qn, qt, h, coord)
% Generates equivalent load vector for a triangular element
% side = side over which the load is specified
% qn, qt = load components in the normal and the tangential direction
% h = thickness
% coord = coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
switch (side)
case 1
    L=sqrt((x2-x1)^2+(y2-y1)^2);
    nx=(y2-y1)/L; ny=-(x2-x1)/L;
    qx = nx*qn - ny*qt;
    qy = ny*qn + nx*qt;
    rq = h*L/2 * [qx; qy; qx; qy; 0; 0];
case 2
    L=sqrt((x2-x3)^2+(y2-y3)^2);
    nx=(y3-y2)/L; ny=-(x3-x2)/L;
    qx = nx*qn - ny*qt;
    qy = ny*qn + nx*qt;
    rq = h*L/2 * [0; 0; qx; qy; qx; qy];
```

```
case 3
    L=sqrt((x3-x1)^2+(y3-y1)^2);
    nx=(y1-y3)/L; ny=-(x1-x3)/L;
    qx = nx*qn - ny*qt;
    qy = ny*qn + nx*qt;
    rq = h*L/2 * [qx; qy; 0; 0; qx; qy];
end
```

Using these functions finite element equations for any triangular element for a plane stress problem can easily be written. As an example we use these functions to develop matrices for the element number 2 in the finite element model of the notched beam. The element is connected between nodes 4, 7 and 11. There is an applied load in the negative outer normal direction ($q_n = -50$ and $q_t = 0$) on side 3 of the element. With $E = 3000000$, $\nu = 0.2$ and $h = 4$ the $\boldsymbol{k}$ matrix and the equivalent nodal load vector $\boldsymbol{r}_q$ for the element are computed as follows.

MatlabFiles\Chap1\PlaneStressElementEx1.m

```
% Plane stress element equations
nodes = [0, 5; 0, 22/3; 0, 29/3; 0, 12; 3, 5; 4, 22/3;
    5, 29/3; 6, 0; 6, 5/2; 6, 5; 6, 12; 8, 22/3; 10, 29/3;
    23/2, 17/3; 12, 12; 15, 4; 17, 53/6; 20, 0;45/2, 12;
    24, 8; 80/3, 4; 33, 12; 100/3, 8; 37,0; 40, 12; 121/3, 4;
    131/3, 8; 47, 12; 54, 0; 54, 4; 54, 8; 54, 12];
k = PlaneStressTriElement(3000000, .2, 4, nodes([4 7 11],:))
rq = PlaneStressTriLoad (3, -50, 0, 4, nodes([4 7 11],:))
```

```
>> PlaneStressElementEx1

k =

  1.0e+007 *

    0.2609   -0.0625   -0.1071    0.1250   -0.1538   -0.0625
   -0.0625    0.1419    0.2500   -0.2679   -0.1875    0.1260
   -0.1071    0.2500    0.6429         0   -0.5357   -0.2500
    0.1250   -0.2679         0    1.6071   -0.1250   -1.3393
   -0.1538   -0.1875   -0.5357   -0.1250    0.6895    0.3125
   -0.0625    0.1260   -0.2500   -1.3393    0.3125    1.2133


rq =

     0
  -600
     0
     0
     0
  -600
```

Computer Implementation 1.4 *(Matlab)*

The assembly process is conceptually straight-forward but is clearly tedious and thus error prone when performing computations by hand. Fortunately it is fairly easy to establish a *Mathematica* or *Matlab* based procedure to assemble equations for any finite element model. The process in *Matlab* is illustrated here. The global system is 10×10. We start the process by defining a 10×10 matrix $\boldsymbol{K}$ and a 10×1 vector $\boldsymbol{R}$ using the Table command as follows.

MatlabFiles\Chap1\AssemblyEx.m

The global system is 10×10. We start the process by defining a 10×10 matrix $\boldsymbol{K}$ and a 10×1 vector $\boldsymbol{R}$ using the Table command as follows.

```
K=zeros(10); R = zeros(10,1);
```

Consider assembly of element 1 whose $\boldsymbol{k}$ matrix and $\boldsymbol{r}$ vector are as follows.

```
k = [111,201,301; 201,222,232; 301,232,333]
r = [11; 12; 13]
```

This element contributes to 1, 2 and 5 degrees of freedom. We define a vector lm (element assembly location vector) to indicate the degrees of freedom to which this element contributes.

```
lm = [1, 2, 5]
```

For assembling r into the global R vector, the appropriate locations are those given in the lm vector. Thus we must extracts elements 1, 2 and 5 of the R vector and add the **r** vector to them. The *Matlab* syntax R(lm) accomplishes the task of extracting the required elements. Thus we can assemble r into global R as follows.

```
R(lm) = R(lm) + r
>>
R =

    11
    12
     0
     0
    13
     0
     0
     0
     0
     0
```

For assembling k into the global K, the appropriate locations are combinations of entries given in the lm vector. *Matlab* generates these combinations automatically if arguments for extracting elements are two lists. Thus we can assemble k into global K as follows.

```
K(lm, lm) = K(lm, lm) + k
>>
K =

   111   201     0     0   301     0     0     0     0     0
   201   222     0     0   232     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   301   232     0     0   333     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Now consider the assembly of element 2 whose $\boldsymbol{k}$ matrix and $\boldsymbol{r}$ vector are as follows.

```
k = [77,80,90; 80,88,100; 90,100,99]
r = [21; 22; 23]
```

This element contributes to 2, 6 and 5 degrees of freedom and thus we define the lm vector for this element as follows.

```
lm = [2, 6, 5]
```

The assembly of r into global R is as follows.

```
R(lm) = R(lm) + r
>>
R =

    11
    33
     0
     0
    36
    22
     0
     0
     0
     0
```

The assembly of k into global K is as follows.

```
K(lm, lm) = K(lm, lm) + k
>>

K =

   111   201     0     0   301     0     0     0     0     0
   201   299     0     0   322    80     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   301   322     0     0   432   100     0     0     0     0
     0    80     0     0   100    88     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Clearly all elements can easily be assembled using this procedure. As will be illustrated in later examples, the process can be streamlined even further by using the *for* loop function in *Matlab*.

Computer Implementation 1.5 *(Matlab)*

Use *Matlab* to develop element equations and assemble them to form global equations for five bar plane truss. The complete procedure, with data in N-mm units, is as follows.

MatlabFiles\Chap1\TrussAssemblyEx.m

```
% Truss Assembly Example
e1=200*10^3; e2=70*10^3; a1=40*100; a2=30*100; a3=20*100;
P = 150*10^3;
nodes = 1000*[0, 0; 1.5, 3.5; 0, 5; 5, 5];
K=zeros(8);
% Generate stiffness matrix for each element and assemble it.
```

```
k=PlaneTrussElement(e1, a1, nodes([1 2],:));
lm=[1, 2, 3, 4];
K(lm, lm) = K(lm, lm) + k;

k=PlaneTrussElement(e1, a1, nodes([2 4],:));
lm=[3, 4, 7, 8];
K(lm, lm) = K(lm, lm) + k;

k=PlaneTrussElement(e1, a2, nodes([1 3],:));
lm=[1, 2, 5, 6];
K(lm, lm) = K(lm, lm) + k;

k=PlaneTrussElement(e1, a2, nodes([3 4],:));
lm=[5, 6, 7, 8];
K(lm, lm) = K(lm, lm) + k;

k=PlaneTrussElement(e2, a3, nodes([2 3],:));
lm=[3, 4, 5, 6];
K(lm, lm) = K(lm, lm) + k

% Define the load vector
R = zeros(8,1); R(4)=-P
```

For each element the PlaneTrussElement function defined in *Matlab* Implementation 1.1 is first used to generate element stiffness matrix, k. From the degrees of freedom the assembly location vector lm for the element is then defined. The assembly is then carried out by the statement K(lm, lm) = K(lm, lm) + k. To conserve space printing of most intermediate results is suppressed by ending statements with a semicolon. If desired, the intermediate results can be seen by executing statements with the semicolon removed. The global load vector is written directly.

```
>>

K =

  1.0e+005 *

  Columns 1 through 7

    0.3260    0.7607   -0.3260   -0.7607         0         0         0
    0.7607    2.9749   -0.7607   -1.7749         0   -1.2000         0
   -0.3260   -0.7607    2.4309    1.1914   -0.3300    0.3300   -1.7749
   -0.7607   -1.7749    1.1914    2.4309    0.3300   -0.3300   -0.7607
         0         0   -0.3300    0.3300    1.5300   -0.3300   -1.2000
         0   -1.2000    0.3300   -0.3300   -0.3300    1.5300         0
         0         0   -1.7749   -0.7607   -1.2000         0    2.9749
         0         0   -0.7607   -0.3260         0         0    0.7607

  Column 8

         0
         0
   -0.7607
   -0.3260
         0
         0
    0.7607
    0.3260
```

```
R =

           0
           0
           0
     -150000
           0
           0
           0
           0
```

Computer Implementation 1.6 *(Matlab)*

MatlabFiles\Chap1\HeatAssemblyEx.m

```
% Heat Assembly Example
kx=1.4; ky=1.4; Q=0;
nodes=[0,0; 20,0; 20,30; 0,10; 10,10]/100;
lmm =[1,2,5; 2,3,5; 3,4,5; 1,5,4];

K=zeros(5); R = zeros(5,1);
% Generate equations for each element and assemble them.
for i=1:4
    lm = lmm(i,:);
    [k, r] = HeatTriElement(kx, ky, Q, nodes(lm,:));
    K(lm, lm) = K(lm, lm) + k;
    R(lm) = R(lm) + r;
end
% Add the term beacuse of convection on side 1 of element 2
h=27;Tinf=20; lm = lmm(2,:);
[kh, rh] = ConvectionTerm(1,h,Tinf,nodes(lm,:));
K(lm, lm) = K(lm, lm) + kh
R(lm) = R(lm) + rh
```

From the node numbering shown in the figure the assembly location vectors for all elements are defined. The HeatTriElement function defined in an earlier *Matlab* Implementation is first used to generate element matrices for all elements in the model. Since all elements have the same thermal properties, and the elements nodes are same as the lm vectors established already, it is convenient to use for loop to compute element $\boldsymbol{k}_k$ and $\boldsymbol{r}_Q$. The element 2 has convection condition on side 1 with $h = 27$ and $T_\infty = 20$. Thus we must add $\boldsymbol{k}_h$ and $\boldsymbol{r}_h$ terms to the second element equations.

```
>> HeatAssemblyEx

K =

    1.4000         0         0   -0.7000   -0.7000
         0    4.5667    1.5833         0   -2.1000
         0    1.5833    3.5167    0.3500   -1.4000
   -0.7000         0    0.3500    3.1500   -2.8000
   -0.7000   -2.1000   -1.4000   -2.8000    7.0000


R =

     0
    81
    81
```

```
         0
         0
```

Computer Implementation 1.7 *(Matlab)*

Use *Matlab* to develop element equations and assemble them to form global equations for the cantilever bracket.

MatlabFiles\Chap1\PlaneStressAssemblyEx.m

```
% Plane Stress Assembly Example
e = 10^4; nu = 0.2; h=0.25; q=-20;
nodes=[0,0; 0,2; 2,0; 2,3/2; 4,0; 4,1];
conn = [1,3,4; 4,2,1; 3,5,6; 6,4,3];
lmm = [1,2,5,6,7,8; 7,8,3,4,1,2;
    5,6,9,10,11,12; 11,12,7,8,5,6];

K=zeros(12); R = zeros(12,1);
% Generate equations for each element and assemble them.
for i=1:4
    con = conn(i,:);
    lm = lmm(i,:);
    k = PlaneStressTriElement(e, nu, h, nodes(con,:));
    K(lm, lm) = K(lm, lm) + k;
end
% Define the nodal load vector
con = conn(2,:);
lm = lmm(2,:);
rq = PlaneStressTriLoad(1,q,0,h,nodes(con,:));
R(lm) = R(lm) + rq;

con = conn(4,:);
lm = lmm(4,:);
rq = PlaneStressTriLoad(1,q,0,h,nodes(con,:));
R(lm) = R(lm) + rq;
K
R
```

The PlaneStressTriElement function defined in *Matlab* Implementation 1.3 is used to generate element matrices for all elements in the model. A normal pressure is applied to side 1 of elements 2 and 4. The pressure acts in the direction opposite to the outer normal to the surface and therefore it must be assigned a negative sign. For the other two elements the r vector is all zeros. From the node numbering shown in the figure, and with two degree of freedom per node, the assembly location vectors for all elements are defined. The assembly can now be carried out easily by using the for loop.

```
>> PlaneStressAssemblyEx

K =

  1.0e+003 *

  Columns 1 through 7

    1.5788    0.1953   -0.2767    0.3255   -0.9766    0.2604   -0.3255
    0.1953    1.7253    0.0651   -1.2044    0.5208   -0.3906   -0.7813
   -0.2767    0.0651    1.2533   -0.5859         0         0   -0.9766
    0.3255   -1.2044   -0.5859    1.5951         0         0    0.2604
   -0.9766    0.5208         0         0    3.1250   -0.5208   -1.1719
    0.2604   -0.3906         0         0   -0.5208    4.1667    0.5208
   -0.3255   -0.7813   -0.9766    0.2604   -1.1719    0.5208    3.1250
   -0.7813   -0.1302    0.5208   -0.3906    0.5208   -3.3854   -0.5208
         0         0         0         0   -0.6510    0.5208         0
         0         0         0         0    0.2604   -0.2604         0
         0         0         0         0   -0.3255   -0.7813   -0.6510
         0         0         0         0   -0.7813   -0.1302    0.5208

  Columns 8 through 12

   -0.7813         0         0         0         0
   -0.1302         0         0         0         0
    0.5208         0         0         0         0
   -0.3906         0         0         0         0
    0.5208   -0.6510    0.2604   -0.3255   -0.7813
   -3.3854    0.5208   -0.2604   -0.7813   -0.1302
   -0.5208         0         0   -0.6510    0.5208
    4.1667         0         0    0.2604   -0.2604
         0    1.6927   -0.7813   -1.0417    0.2604
         0   -0.7813    2.8646    0.5208   -2.6042
    0.2604   -1.0417    0.5208    2.0182         0
   -0.2604    0.2604   -2.6042         0    2.9948


R =

         0
         0
   -1.2500
   -5.0000
         0
         0
   -2.5000
  -10.0000
         0
         0
   -1.2500
   -5.0000
```

Computer Implementation 1.8 *(Matlab)*

Computer Implementation 1.9 *(Matlab)*

The manipulations for imposing essential boundary conditions and final solution for nodal unknowns can easily be achieved using *Matlab* by defining the following NodalSoln function.

MatlabFiles\Chap1\NodalSoln.m

```
function [d, rf] = NodalSoln(K, R, debc, ebcVals)
% [nd, rf] = NodalSoln(K, R, debc, ebcVals)
% Computes nodal solution and reactions
% K = global coefficient matrix
% R = global right hand side vector
% debc = list of degrees of freedom with specified values
% ebcVals = specified values
dof = length(R);
df = setdiff(1:dof, debc);
Kf = K(df, df);
```

```
Rf = R(df) - K(df, debc)*ebcVals;
dfVals = Kf\Rf;
d = zeros(dof,1);
d(debc) = ebcVals;
d(df) = dfVals;
rf = K(debc,:)*d - R(debc);
```

Two lists are established to contain degrees of freedom where essential boundary conditions are to be imposed and their corresponding values. The list of degrees of freedom to be retained are obtained by taking complement of the list of all degrees of freedom with the debc. Using the list of remaining degrees of freedom the final system of equations can now be established easily. The solution for nodal unknowns can now be obtained by using the `\` operation in *Matlab*. The complete vector of nodal values, in the original order established by the node numbering, is obtained by combining these values with those specified as essential boundary conditions. This function is used in almost all Matlab implementations presented in later chapters.

MatlabFiles\Chap1\EssentialBCEx.m

```
% Incorporate essential boundary conditions
K=[1000,10,20,30,40,50; 10,2000,21,31,41,51;
    20,21,3000,32,42,52; 30,31,32,4000,43,53;
    40,41,42,43,5000,54; 50,51,52,53,54,6000];
R=[100; 110; 120; 130; 140; 150];
[d, reactions] = NodalSoln(K, R, [1, 4, 5], [5; -7; 0])
```

```
>> EssentialBCEx

d =

    5.0000
    0.1366
    0.0796
   -7.0000
         0
    0.0433


reactions =

  1.0e+004 *

    0.4695
   -2.7971
   -0.0230
```

Computer Implementation 1.10 *(Matlab)*

The solution for each element for a plane truss can easily be generated by writing a simple function in *Matlab*. The following *PlaneTrussResults* function returns the axial strain, axial stress, and axial force for each element.

MatlabFiles\Chap1\PlaneTrussResults.m

```
function results = PlaneTrussResults(e, A, coord, disps)
% results = PlaneTrussResults(e, A, coord, disps)
% Compute plane truss element results
% e = modulus of elasticity
% A = Area of cross-section
% coord = coordinates at the element ends
% disps = displacements at element ends
% The output quantities are eps = axial strain
% sigma = axial stress and force = axial force.

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
L=sqrt((x2-x1)^2+(y2-y1)^2);
ls=(x2-x1)/L; ms=(y2-y1)/L;
T=[ls,ms,0,0; 0,0,ls,ms];
d = T*disps;
eps= (d(2)-d(1))/L;
sigma = e.*eps;
force = sigma.*A;
results=[eps, sigma, force];
```

Computer Implementation 1.11 *(Matlab)* *Complete solution of a plane truss*

By combining procedures discussed in earlier implementations, here we present a complete *Matlab* based solution for the five bar truss. This implementation can be used as template to analyze any other plane truss. The global matrices K and R are generated first using statements discussed earlier. To conserve space, printing of individual element matrices is suppressed by using semicolon at the end of each line.

MatlabFiles\Chap1\FiveBarTrussEx.m

```
% Five bar truss example
e1=200*10^3; e2=70*10^3; a1=40*100; a2=30*100; a3=20*100;
P = 150*10^3;
nodes = 1000*[0, 0; 1.5, 3.5; 0, 5; 5, 5];
conn = [1,2; 2,4; 1,3; 3,4; 2,3];
lmm = [1,2,3,4; 3, 4, 7, 8; 1, 2, 5, 6; 5, 6, 7, 8; 3, 4, 5, 6];
K=zeros(8);
% Generate stiffness matrix for each element and assemble it.
for i=1:2
    lm=lmm(i,:);
    con=conn(i,:);
    k=PlaneTrussElement(e1, a1, nodes(con,:));
    K(lm, lm) = K(lm, lm) + k
end
for i=3:4
    lm=lmm(i,:);
    con=conn(i,:);
    k=PlaneTrussElement(e1, a2, nodes(con,:));
    K(lm, lm) = K(lm, lm) + k
end

lm=lmm(5,:); con=conn(5,:);
k=PlaneTrussElement(e2, a3, nodes(con,:));
K(lm, lm) = K(lm, lm) + k

% Define the load vector
R = zeros(8,1); R(4)=-P

% Nodal solution and reactions
[d, reactions] = NodalSoln(K, R, [1,2,7,8], zeros(4,1))
results=[];
for i=1:2
    results = [results; PlaneTrussResults(e1, a1, ...
            nodes(conn(i,:),:), d(lmm(i,:)))];
end
for i=3:4
    results = [results; PlaneTrussResults(e1, a2, ...
            nodes(conn(i,:),:), d(lmm(i,:)))];
end
format short g
results = [results; PlaneTrussResults(e2, a3, ...
        nodes(conn(5,:),:), d(lmm(5,:)))]


>> FiveBarTrussEx

K =

  Columns 1 through 6

        32600        76067       -32600       -76067            0
0
        76067  2.9749e+005       -76067 -1.7749e+005            0
-1.2e+005
       -32600       -76067  2.4309e+005  1.1914e+005       -32998
32998
       -76067 -1.7749e+005  1.1914e+005  2.4309e+005        32998
-32998
            0            0       -32998        32998    1.53e+005
-32998
            0    -1.2e+005        32998       -32998       -32998
1.53e+005
            0            0 -1.7749e+005       -76067    -1.2e+005
0
            0            0       -76067       -32600            0
0

  Columns 7 through 8

            0            0
            0            0
 -1.7749e+005       -76067
       -76067       -32600
    -1.2e+005            0
            0            0
  2.9749e+005        76067
        76067        32600


R =

           0
           0
           0
     -150000
           0
           0
           0
           0
```

```
d =

            0
            0
      0.53895
     -0.95306
       0.2647
      -0.2647
            0
            0


reactions =

        54927
   1.5993e+005
       -54927
      -9926.7


results =

   -0.0001743      -34.859 -1.3944e+005
   -3.15e-005      -6.2999       -25200
 -5.2941e-005      -10.588       -31764
 -5.2941e-005      -10.588       -31764
   0.00032087       22.461        44922
```

Computer Implementation 1.12 *(Matlab)*

The solution for each element for a heat flow problem can easily be generated by writing a simple function in *Matlab*. The following *HeatTriResults* function returns the temperature $T(x, y)$ and its x and y derivatives for each element at the center of each element.

MatlabFiles\Chap1\HeatTriResults.m

```
function results = HeatTriResults(coord, tn)
% results = HeatTriResults(coord, tn)
% Computes element solution for a heat flow triangular element
% coord = nodal coordinates
% tn = nodal temperatures
% Output variables are T and its x and y derivatives at element center

x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
x=(x1+x2+x3)/3; y=(y1+y2+y3)/3;
b1 = y2 - y3; b2 = y3 - y1; b3 = y1 - y2;
c1 = x3 - x2; c2 = x1 - x3; c3 = x2 - x1;
f1 = x2*y3 - x3*y2; f2 = x3*y1 - x1*y3; f3 = x1*y2 - x2*y1;
A = (f1 + f2 + f3)/2;
n = [f1 + x*b1 + y*c1, f2 + x*b2 + y*c2, f3 + x*b3 + y*c3]/(2*A);
T = n*tn; dTdx=[b1, b2, b3]*tn/(2*A); dTdy=[c1, c2, c3]*tn/(2*A);
results=[T, dTdx, dTdy];
```

Computer Implementation 1.13 ***(Matlab)*** *Complete solution of a heat flow problem*

By combining procedures discussed in earlier implementations, here we present a complete *Matlab* based solution for the heat flow model. This implementation can be used as template to analyze any other two dimensional heat flow problem.

MatlabFiles\Chap1\SquareDuctHeatEx.m

```
% Heat flow through a square duct example
kx=1.4; ky=1.4; Q=0;
nodes=[0,0; 20,0; 20,30; 0,10; 10,10]/100;
lmm =[1,2,5; 2,3,5; 3,4,5; 1,5,4];

K=zeros(5); R = zeros(5,1);
% Generate equations for each element and assemble them.
for i=1:4
    lm = lmm(i,:);
    [k, r] = HeatTriElement(kx, ky, Q, nodes(lm,:));
    K(lm, lm) = K(lm, lm) + k;
    R(lm) = R(lm) + r;
end
% Add the term beacuse of convection on side 1 of element 2
h=27;Tinf=20; lm = lmm(2,:);
[kh, rh] = ConvectionTerm(1,h,Tinf,nodes(lm,:));
K(lm, lm) = K(lm, lm) + kh
R(lm) = R(lm) + rh

% Nodal solution and reactions
[d, reactions] = NodalSoln(K, R, [1,4], [300; 300])
results=[];
for i=1:4
    results = [results; HeatTriResults(nodes(lmm(i,:),:), d(lmm(i,:)))];
end
results
```

```
>> SquareDuctHeatEx

K =

          1.4            0            0         -0.7         -0.7
            0       4.5667       1.5833            0         -2.1
            0       1.5833       3.5167         0.35         -1.4
         -0.7            0         0.35         3.15         -2.8
         -0.7         -2.1         -1.4         -2.8            7


R =

     0
    81
    81
     0
     0


d =
            300
         93.547
         23.844
            300
         182.83


reactions =

         82.017
         231.41


results =

         192.13      -1032.3      -139.41
         100.07      -1125.2      -232.34
         168.89      -1171.7      -209.11
         260.94      -1171.7            0
```

Computer Implementation 1.14 *(Matlab)*

The solution for each element for a plane stress problem can easily be generated by writing a simple function in *Matlab*. The following *PlaneStressTriResults* function returns strain and stress components, principal stresses, and equivalent von Mises stress for each element.

MatlabFiles\Chap1\PlaneStressTriResults.m

```
function se = PlaneStressTriResults(e, nu, coord, dn)
% se = PlaneStressTriResults(e, nu, coord, dn)
% Computes element solution for a plane stress triangular element
% coord = nodal coordinates
% e = modulus of elasticity
% nu = Poisson's ratio
% dn = nodal displacements
% Following are the output variables are at element center
% {strains, stresses, principal stresses, effective stress}
x1=coord(1,1); y1=coord(1,2);
x2=coord(2,1); y2=coord(2,2);
x3=coord(3,1); y3=coord(3,2);
x=(x1+x2+x3)/3; y=(y1+y2+y3)/3;
b1 = y2 - y3; b2 = y3 - y1; b3 = y1 - y2;
c1 = x3 - x2; c2 = x1 - x3; c3 = x2 - x1;
f1 = x2*y3 - x3*y2; f2 = x3*y1 - x1*y3; f3 = x1*y2 - x2*y1;
A = (f1 + f2 + f3)/2;
C = e/(1 - nu^2)*[1, nu, 0; nu, 1, 0; 0, 0, (1 - nu)/2];
B = [b1, 0, c1; 0, c1, b1; b2, 0, c2; 0, c2, b2;
    b3, 0, c3; 0,c3, b3]/(2*A);
eps = B'*dn
sig = C*eps
sx = sig(1); sy= sig(2); sxy=sig(3);
PrincipalStresses = eig([sx,sxy; sxy,sy])
se = sqrt((sx - sy)^2 + sy^2 + sx^2 + 6*sxy^2)/sqrt(2);
```

Computer Implementation 1.15 ***(Matlab)*** *Complete solution of a plane stress problem*

By combining procedures discussed in earlier implementations, here we present a complete *Matlab* based solution for the plane stress bracket model. This implementation can be used as template to analyze any other plane stress analysis problem.

MatlabFiles\Chap1\PlaneStressBracketEx.m

```
% Plane stress analysis of bracket
e = 10^4; nu = 0.2; h=0.25; q=-20;
nodes=[0,0; 0,2; 2,0; 2,3/2; 4,0; 4,1];
conn = [1,3,4; 4,2,1; 3,5,6; 6,4,3];
lmm = [1,2,5,6,7,8; 7,8,3,4,1,2;
    5,6,9,10,11,12; 11,12,7,8,5,6];

K=zeros(12); R = zeros(12,1);
% Generate equations for each element and assemble them.
for i=1:4
    con = conn(i,:);
    lm = lmm(i,:);
    k = PlaneStressTriElement(e, nu, h, nodes(con,:));
    K(lm, lm) = K(lm, lm) + k;
end
% Define the nodal load vector
con = conn(2,:);
lm = lmm(2,:);
rq = PlaneStressTriLoad(1,q,0,h,nodes(con,:));
R(lm) = R(lm) + rq;

con = conn(4,:);
lm = lmm(4,:);
rq = PlaneStressTriLoad(1,q,0,h,nodes(con,:));
R(lm) = R(lm) + rq;

% Nodal solution and reactions
[d, reactions] = NodalSoln(K, R, [1,2,3,4], zeros(4,1))
for i=1:4
    fprintf(1,'Results for element %3.0g \n',i)
    EffectiveStress=PlaneStressTriResults(e, nu, ...
        nodes(conn(i,:),:), d(lmm(i,:)))
end


>> PlaneStressBracketEx

d =

            0
            0
            0
            0
    -0.010355
     -0.02553
    0.0047277
    -0.024736
    -0.013139
    -0.055493
```

```
   8.389e-005
    -0.055566


reactions =

        21.25
       4.1065
       -16.25
       15.894

Results for element   1

eps =

   -0.0051776
   0.00052936
   -0.0027096


sig =

      -52.831
      -5.2726
       -11.29


PrincipalStresses =

      -55.375
      -2.7286


EffectiveStress =

       54.062

Results for element   2

eps =

    0.0023638
            0
    -0.012368


sig =

       24.623
       4.9246
      -51.533


PrincipalStresses =

      -37.691
       67.239
```

```
EffectiveStress =

       92.066

Results for element   3

eps =

   -0.0013921
 -7.3267e-005
   -0.0017584


sig =

      -14.653
      -3.6633
      -7.3267


PrincipalStresses =

      -18.317
 -2.7978e-014


EffectiveStress =

       18.317

Results for element   4

eps =

   0.00019194
   0.00052936
   -0.0052277


sig =

       3.1022
       5.9141
      -21.782


PrincipalStresses =

      -17.319
       26.336


EffectiveStress =

       38.074
```