CHAPTER FIVE

Two Dimensional Elements

Computer Implementation 5.1 *(Matlab)*

The analysis of two dimensional bounadry value problems using rectangular elements can be performed conveniently by writing three *Matlab* functions, one for defining the element $\boldsymbol{k}_k + \boldsymbol{k}_p$ and $\boldsymbol{r}_q$ vectors, one for evaluating natural boundary terms, and the third for computing the element solution.

MatlabFiles\Chap5\BVPRectElement.m

```
function [ke, rq] = BVPRectElement(kx, ky, p, q, coord)
% [ke, rq] = BVPRectElement(kx, ky, p, q, coord)
% Generates for a rectangular element for 2d BVP
% kx, ky, p, q = parameters defining the BVP
% 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);
```

```
x4=coord(4,1); y4=coord(4,2);
a = abs((x2 - x1))/2; b = abs((y4 - y2))/2;
kk = (kx*b)/(6*a)*[2, -2, -1, 1; -2, 2, 1, -1;
   -1, 1, 2, -2; 1, -1, -2, 2] + ...
   (ky*a)/(6*b)*[2, 1, -1, -2; 1, 2, -2, -1;
   -1, -2, 2, 1; -2, -1,1, 2];
kp = -((p*a*b)/9)*[4, 2,1, 2; 2, 4, 2, 1; 1, 2, 4, 2; 2, 1, 2, 4];
ke = kk + kp;
rq = a*b*q*[1; 1; 1; 1];
```

MatlabFiles\Chap5\BVPRectNBCTerm.m

```
function [ka, rb] = BVPRectNBCTerm(side, alpha, beta, coord)
% [ka, rb] = BVPRectNBCTerm(side, alpha, beta, coord)
% Generates kalpha and rbeta when NBC is specified along a side
% side = side over which the NBC is specified
% alpha and beta = coefficients specifying the NBC
% 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);
x4=coord(4,1); y4=coord(4,2);
ka = zeros(4); rb = zeros(4,1);
switch (side)
case 1
   lm=[1,2]; L=abs(x2-x1);
case 2
   lm=[2,3]; L=abs(y3-y2);
case 3
   lm=[3,4]; L=abs(x4-x3);
case 4
   lm=[4,1]; L=abs(y4-y1);
end
ka(lm, lm) = -(1/3)*alpha*L/2*[2, 1; 1, 2];
rb(lm) = beta*L/2*[1; 1];
```

MatlabFiles\Chap5\BVPRectResults.m

```
function results = BVPRectResults(coord, dn)
% results = BVPRectResults(coord, dn)
% Computes element solution for a rectangular element for 2D BVP
% coord = nodal coordinates
% dn = nodal solution
% Output variables are u 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);
x4=coord(4,1); y4=coord(4,2);
s = 0; t = 0;
a = abs((x2 - x1))/2; b = abs((y4 - y2))/2;
u = 1/(4*a*b)*[(a - s)*(b - t), (a + s)*(b - t), ...
      (a + s)*(b + t), (a - s)*(b + t)]*dn;
dudx= 1/(4*a*b)*[-b + t, b - t, b + t, -b - t]*dn;
dudy = 1/(4*a*b)*[-a + s, -a - s, a + s, a - s]*dn;
results=[u, dudx, dudy];
```

Using these functions now we consider solution of the heat flow problem.

MatlabFiles\Chap5\LShapedHeatEx.m

```
% Heat flow through an L-shaped body
h = 55; tf = 20; htf = h*tf;
kx = 45;  ky = 45; Q = 5*10^6; ql = 8000; t0 = 110;
nodes = 1.5/100*[0, 2; 1,2; 2, 2; 0, 1; 1, 1; 2, 1; 3, 1; 4, 1;
   0, 0; 1, 0; 2, 0; 3, 0; 4, 0];
lmm = [9, 10, 5, 4; 4, 5, 2, 1; 10, 11, 6, 5;
   5, 6, 3, 2; 11, 12, 7, 6; 12, 13, 8, 7];
debc = [9:13]; ebcVals=t0*ones(length(debc),1);
dof=length(nodes); elems=size(lmm,1);
K=zeros(dof); R = zeros(dof,1);
% Generate equations for each element and assemble them.
for i=1:elems
   lm = lmm(i,:);
   [k, r] = BVPRectElement(kx, ky, 0, Q, nodes(lm,:));
   K(lm, lm) = K(lm, lm) + k;
   R(lm) = R(lm) + r;
end
% Compute and assemble NBC contributions
lm = lmm(1,:);
[k, r] = BVPRectNBCTerm(4, 0, ql, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(2,:);
[k, r] = BVPRectNBCTerm(4, 0, ql, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPRectNBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(4,:);
```

```
[k, r] = BVPRectNBCTerm(2, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPRectNBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(5,:);
[k, r] = BVPRectNBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(6,:);
[k, r] = BVPRectNBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

% Nodal solution
d = NodalSoln(K, R, debc, ebcVals)
results=[];
for i=1:elems
   results = [results; BVPRectResults(nodes(lmm(i,:),:), d(lmm(i,:)))];
end
results
```

```
>> LShapedHeatEx

d =

  154.9620
  151.2283
  148.6731
  145.4325
  142.5208
  134.8705
  122.4359
  121.0878
  110.0000
  110.0000
  110.0000
  110.0000
  110.0000


results =

  1.0e+003 *
```

```
0.1270   -0.0971    2.2651
0.1485   -0.2215    0.6079
0.1243   -0.2550    1.9130
0.1443   -0.3402    0.7503
0.1193   -0.4145    1.2435
0.1159   -0.0449    0.7841
```

Computer Implementation 5.2 *(Matlab)*

The following functions implement the 8 node rectangular element in *Matlab*. The first function defines the element $\boldsymbol{k}_k + \boldsymbol{k}_p$ and $\boldsymbol{r}_q$ vectors, the second function evaluates terms resulting from any specified natural boundary conditions, and the third function computes the element solution.

MatlabFiles\Chap5\BVPRect8Element.m

```
function [ke, rq] = BVPRect8Element(kx, ky, p, q, coord)
% [ke, rq] = BVPRect8Element(kx, ky, p, q, coord)
% 8 node rectangular element for 2d BVP
% kx, ky, p, q = parameters defining the BVP
% coord = coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x3=coord(3,1); y7=coord(7,2);
a = abs(x3 - x1)/2; b = abs(y7 - y1)/2;
kk = [(26*b*kx)/(45*a) + (26*a*ky)/(45*b), ...
      (-8*b*kx)/(9*a) + (a*ky)/(15*b), ...
      (14*b*kx)/(45*a) + (17*a*ky)/(90*b), ...
      -(b*kx)/(15*a) - (4*a*ky)/(9*b), ...
      (23*b*kx)/(90*a) + (23*a*ky)/(90*b), ...
      (-4*b*kx)/(9*a) - (a*ky)/(15*b), ...
      (17*b*kx)/(90*a) + (14*a*ky)/(45*b), ...
      (b*kx)/(15*a) - (8*a*ky)/(9*b);
   (-8*b*kx)/(9*a) + (a*ky)/(15*b), ...
      (16*b*kx)/(9*a) + (8*a*ky)/(15*b), ...
      (-8*b*kx)/(9*a) + (a*ky)/(15*b), 0, ...
      (-4*b*kx)/(9*a) - (a*ky)/(15*b), ...
      (8*b*kx)/(9*a) - (8*a*ky)/(15*b), ...
      (-4*b*kx)/(9*a) - (a*ky)/(15*b), 0;
   (14*b*kx)/(45*a) + (17*a*ky)/(90*b),...
      (-8*b*kx)/(9*a) + (a*ky)/(15*b), ...
      (26*b*kx)/(45*a) + (26*a*ky)/(45*b), ...
      (b*kx)/(15*a) - (8*a*ky)/(9*b), ...
      (17*b*kx)/(90*a) + (14*a*ky)/(45*b),...
      (-4*b*kx)/(9*a) - (a*ky)/(15*b), ...
```

```
        (23*b*kx)/(90*a) + (23*a*ky)/(90*b),...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b);
    -(b*kx)/(15*a) - (4*a*ky)/(9*b), 0, ...
        (b*kx)/(15*a) - (8*a*ky)/(9*b), ...
        (8*b*kx)/(15*a) + (16*a*ky)/(9*b),...
        (b*kx)/(15*a) - (8*a*ky)/(9*b), 0,...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b), ...
        (-8*b*kx)/(15*a) + (8*a*ky)/(9*b);
    (23*b*kx)/(90*a) + (23*a*ky)/(90*b),...
        (-4*b*kx)/(9*a) - (a*ky)/(15*b),...
        (17*b*kx)/(90*a) + (14*a*ky)/(45*b),...
        (b*kx)/(15*a) - (8*a*ky)/(9*b), ...
        (26*b*kx)/(45*a) + (26*a*ky)/(45*b),...
        (-8*b*kx)/(9*a) + (a*ky)/(15*b),...
        (14*b*kx)/(45*a) + (17*a*ky)/(90*b),...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b);
    (-4*b*kx)/(9*a) - (a*ky)/(15*b), ...
        (8*b*kx)/(9*a) - (8*a*ky)/(15*b),...
        (-4*b*kx)/(9*a) - (a*ky)/(15*b), 0, ...
        (-8*b*kx)/(9*a) + (a*ky)/(15*b), ...
        (16*b*kx)/(9*a) + (8*a*ky)/(15*b), ...
        (-8*b*kx)/(9*a) + (a*ky)/(15*b), 0;
    (17*b*kx)/(90*a) + (14*a*ky)/(45*b),...
        (-4*b*kx)/(9*a) - (a*ky)/(15*b),...
        (23*b*kx)/(90*a) + (23*a*ky)/(90*b),...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b),...
        (14*b*kx)/(45*a) + (17*a*ky)/(90*b),...
        (-8*b*kx)/(9*a) + (a*ky)/(15*b),...
        (26*b*kx)/(45*a) + (26*a*ky)/(45*b), ...
        (b*kx)/(15*a) - (8*a*ky)/(9*b);
    (b*kx)/(15*a) - (8*a*ky)/(9*b), 0, ...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b),...
        (-8*b*kx)/(15*a) + (8*a*ky)/(9*b),...
        -(b*kx)/(15*a) - (4*a*ky)/(9*b), 0,...
        (b*kx)/(15*a) - (8*a*ky)/(9*b), (8*b*kx)/(15*a) + (16*a*ky)/(9*b)];
kp = [(-(2/15))*a*b*p, (2*a*b*p)/15, (-(2/45))*a*b*p, (8*a*b*p)/45,...
        (-(1/15))*a*b*p, (8*a*b*p)/45, (-(2/45))*a*b*p, (2*a*b*p)/15;
    (2*a*b*p)/15, (-(32/45))*a*b*p, (2*a*b*p)/15, (-(4/9))*a*b*p,...
        (8*a*b*p)/45, (-(16/45))*a*b*p, (8*a*b*p)/45, (-(4/9))*a*b*p;
    (-(2/45))*a*b*p, (2*a*b*p)/15, (-(2/15))*a*b*p, (2*a*b*p)/15, ...
        (-(2/45))*a*b*p, (8*a*b*p)/45, (-(1/15))*a*b*p, (8*a*b*p)/45;
    (8*a*b*p)/45, (-(4/9))*a*b*p, (2*a*b*p)/15, (-(32/45))*a*b*p,...
        (2*a*b*p)/15, (-(4/9))*a*b*p, (8*a*b*p)/45, (-(16/45))*a*b*p;
    (-(1/15))*a*b*p, (8*a*b*p)/45, (-(2/45))*a*b*p, (2*a*b*p)/15,...
        (-(2/15))*a*b*p, (2*a*b*p)/15, (-(2/45))*a*b*p, (8*a*b*p)/45;
    (8*a*b*p)/45, (-(16/45))*a*b*p, (8*a*b*p)/45, (-(4/9))*a*b*p,...
```

```
      (2*a*b*p)/15, (-(32/45))*a*b*p, (2*a*b*p)/15, (-(4/9))*a*b*p;
   (-(2/45))*a*b*p, (8*a*b*p)/45, (-(1/15))*a*b*p, (8*a*b*p)/45,...
      (-(2/45))*a*b*p, (2*a*b*p)/15, (-(2/15))*a*b*p, (2*a*b*p)/15;
   (2*a*b*p)/15, (-(4/9))*a*b*p, (8*a*b*p)/45, (-(16/45))*a*b*p, ...
      (8*a*b*p)/45, (-(4/9))*a*b*p, (2*a*b*p)/15, (-(32/45))*a*b*p];
ke = kk + kp;
rq = [(-(1/3))*a*b*q; (4*a*b*q)/3; (-(1/3))*a*b*q;
   (4*a*b*q)/3; (-(1/3))*a*b*q; (4*a*b*q)/3;
   (-(1/3))*a*b*q; (4*a*b*q)/3];
```

MatlabFiles\Chap5\BVPRect8NBCTerm.m

```
function [ka, rb] = BVPRect8NBCTerm(side, alpha, beta, coord)
% [ka, rb] = BVPRect8NBCTerm(side, alpha, beta, coord)
% 8 node rectangular element for 2d BVP
% Generates kalpha and rbeta when NBC is specified along a side
% side = side over which the NBC is specified
% alpha and beta = coefficients specifying the NBC
% coord = coordinates at the element ends

x1=coord(1,1); y1=coord(1,2);
x3=coord(3,1); y7=coord(7,2);
a = abs(x3 - x1)/2; b = abs(y7 - y1)/2;
switch (side)
case 1
   ka = [-((4*a*alpha)/15), -((2*a*alpha)/15),...
         (a*alpha)/15, 0, 0, 0, 0, 0;
      -((2*a*alpha)/15), -((16*a*alpha)/15),...
         -((2*a*alpha)/15), 0, 0, 0, 0, 0;
      (a*alpha)/15, -((2*a*alpha)/15), ...
         -((4*a*alpha)/15), 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];
   rb = [(a*beta)/3; (4*a*beta)/3; (a*beta)/3;
      0; 0; 0; 0; 0];
case 2
   ka = [0, 0, 0, 0, 0, 0, 0, 0;
      0, 0, 0, 0, 0, 0, 0, 0;
      0, 0, -((4*alpha*b)/15),...
         -((2*alpha*b)/15), (alpha*b)/15, 0, 0, 0;
      0, 0, -((2*alpha*b)/15), -((16*alpha*b)/15), ...
         -((2*alpha*b)/15), 0, 0, 0;
      0, 0, (alpha*b)/15, -((2*alpha*b)/15),...
         -((4*alpha*b)/15), 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];
    rb = [0; 0; (b*beta)/3; (4*b*beta)/3; (b*beta)/3; 0;
        0; 0];
case 3
    ka = [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,
        -((4*a*alpha)/15), -((2*a*alpha)/15),...
            (a*alpha)/15, 0;
        0, 0, 0, 0,-((2*a*alpha)/15), -((16*a*alpha)/15), ...
            -((2*a*alpha)/15), 0;
        0, 0, 0, 0, (a*alpha)/15, -((2*a*alpha)/15),...
            -((4*a*alpha)/15), 0;
        0, 0, 0, 0, 0, 0, 0, 0];
    rb = [0; 0; 0; 0; (a*beta)/3; (4*a*beta)/3;
        (a*beta)/3; 0];
case 4
    ka = [-((4*alpha*b)/15), 0, 0, 0, 0, 0, (alpha*b)/15,...
            -((2*alpha*b)/15);
        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;
        (alpha*b)/15, 0, 0, 0, 0, 0,...
            -((4*alpha*b)/15), -((2*alpha*b)/15);
        -((2*alpha*b)/15), 0, 0, 0, 0, 0,...
            -((2*alpha*b)/15), -((16*alpha*b)/15)];
    rb = [(b*beta)/3; 0; 0; 0; 0; 0; (b*beta)/3;(4*b*beta)/3];
end
```

MatlabFiles\Chap5\BVPRect8Results.m

```
function results = BVPRect8Results(coord, dn)
% results = BVPRect8Results(coord, dn)
% 8 node rectangular element for 2d BVP
% Computes element solution
% coord = nodal coordinates
% dn = nodal solution
% Output variables are u and its x and y derivatives at element center
x1=coord(1,1); y1=coord(1,2);
x3=coord(3,1); y7=coord(7,2);
a = abs(x3 - x1)/2; b = abs(y7 - y1)/2;
```

```
s = 0; t = 0;
u = [-(((a - s)*(b - t)*(b*s + a*(b + t)))/(4*a^2*b^2)),...
      ((a^2 - s^2)*(b - t))/(2*a^2*b),...
      -(((a + s)*(b - t)*(a*(b + t) - b*s))/(4*a^2*b^2)),...
      ((a + s)*(b^2 - t^2))/(2*a*b^2),...
      -(((a + s)*(a*(b - t) - b*s)*(b + t))/(4*a^2*b^2)),...
      ((a^2 - s^2)*(b + t))/(2*a^2*b),...
      -(((a - s)*(b*s + a*(b - t))*(b + t))/(4*a^2*b^2)),...
      ((a - s)*(b^2 - t^2))/(2*a*b^2)]*dn;
dudx= [((b - t)*(2*b*s + a*t))/(4*a^2*b^2), ...
      (s*(t - b))/(a^2*b), -(((b - t)*(a*t - 2*b*s))/(4*a^2*b^2)),...
      (b^2 - t^2)/(2*a*b^2),...
      ((b + t)*(2*b*s + a*t))/(4*a^2*b^2), ...
      -((s*(b + t))/(a^2*b)), ...
      -(((b + t)*(a*t - 2*b*s))/(4*a^2*b^2)), ...
      (t^2 - b^2)/(2*a*b^2)]*dn;
dudy = [((a - s)*(b*s + 2*a*t))/(4*a^2*b^2),...
      (s^2 - a^2)/(2*a^2*b),...
      ((a + s)*(2*a*t - b*s))/(4*a^2*b^2),...
      -(((a + s)*t)/(a*b^2)),...
      ((a + s)*(b*s + 2*a*t))/(4*a^2*b^2),...
      (a^2 - s^2)/(2*a^2*b), ...
      ((a - s)*(2*a*t - b*s))/(4*a^2*b^2),...
      ((s - a)*t)/(a*b^2)]*dn;
results=[u, dudx, dudy];
```

As an example we use these functions to obtain a finite element solution of the Laplace equation considered in Example 5.1 We take advantage of symmetry and model the left-half of the domain using two 8 node elements as shown in Figure 5.16. (The results should obviously be identical to those obtained by using 4 square elements for the entire domain.) There are 13 nodes and thus we have a total of 13 degrees of freedom.

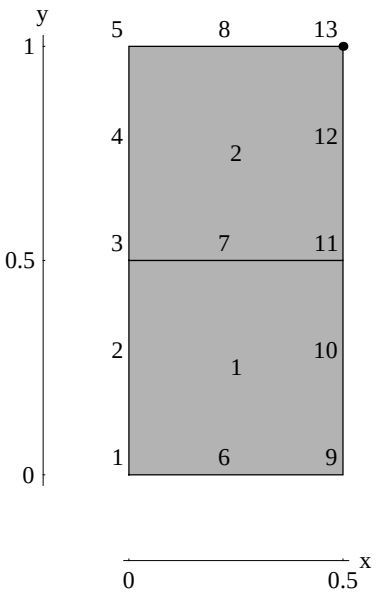

Figure 5.16. Two element model for half of solution domain using 8 node elements

Along line of symmetry (nodes 9 through 13), the normal derivative is 0. From the given essential boundary conditions the following nodal values are known.

Essential boundary conditions {node, value}:

$$\left(\{1,\,0\}\quad\{2,\,0\}\quad\{3,\,0\}\quad\{4,\,0\}\quad\{5,\,0\}\quad\left\{6,\,\tfrac{3}{16}\right\}\quad\{8,\,0\}\quad\left\{9,\,\tfrac{1}{4}\right\}\quad\{13,\,0\}\right)$$

MatlabFiles\Chap5\PoissonEqnEx.m

```
% Solution of poisson equation example using
% two 8 node elements (symmetric left half model)
kx = 1;  ky = 1; p=0; q=0;
nodes=[0,0; 0,1/4; 0,1/2; 0,3/4;
   0,1; 1/4,0; 1/4,1/2; 1/4,1; 1/2,0; 1/2,1/4;
   1/2,1/2; 1/2,3/4; 1/2,1];
lmm=[1,6,9,10,11,7,3,2;
```

```
    3,7,11,12,13,8,5,4];
debc=[1,2,3,4,5,6,8,9,13]';
ebcVals=[0,0,0,0,0,3/16,0,1/4,0]';
dof=length(nodes); elems=size(lmm,1);
K=zeros(dof); R = zeros(dof,1);
% Generate equations for each element and assemble them.
for i=1:elems
    lm = lmm(i,:);
    [k, r] = BVPRect8Element(kx, ky, p, q, nodes(lm,:));
    K(lm, lm) = K(lm, lm) + k;
    R(lm) = R(lm) + r;
end
% Nodal solution
d = NodalSoln(K, R, debc, ebcVals)
results=[];
for i=1:elems
    results = [results; BVPRect8Results(nodes(lmm(i,:),:), d(lmm(i,:)))];
end
results
```

```
>> PoissonEqnEx

d =

         0
         0
         0
         0
         0
    0.1875
    0.0337
         0
    0.2500
    0.1094
    0.0576
    0.0205
         0


results =

    0.0884    0.2189   -0.3076
    0.0127    0.0410   -0.0674
```

In Figure 5.17 the finite element solution at the element centers is compared with the exact solution with five terms in the sum. The solution obtained earlier in Example 5.1 using 4 node elements is also shown for

comparison. Clearly the 8 node element solution is much closer to the exact solution than that obtained using 4 node elements. The solution can obviously be further improved by using more elements.

Even though both finite element models employ the same number of elements, it obviously takes more effort to obtain solution using higher-order (8 node) elements. In fact the computational effort using eight 4 node elements (15 nodes total) is comparable to that of using two 8 node elements (13 nodes total). Since it is easier to derive and program equations for low-order elements, in practice, use of low-order elements is quite common. One simply uses more elements to improve accuracy. However when more accurate solution derivatives are desired generally higher-order elements are preferred over their low-order counterparts. The reason being that the higher-order elements are based on assumed solutions that involve quadratic and higher-order terms and therefore solution derivatives obtained from these elements may be more accurate even with coarse meshes.

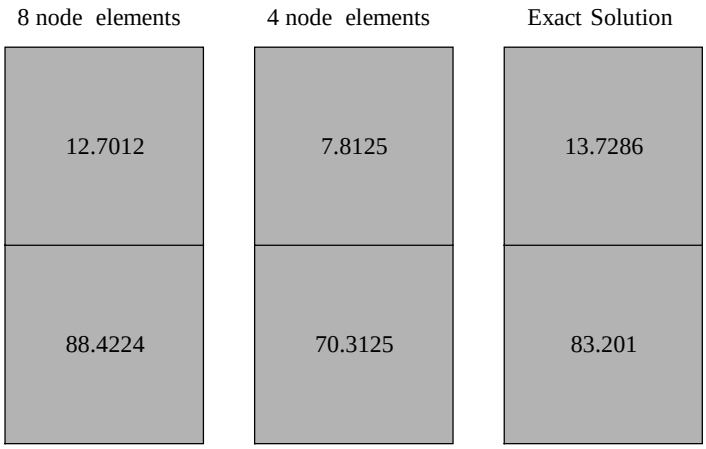

Figure 5.17. Comparison of 4 node and 8 node element solutions with the exact solution $\left(\times 10^{-3}\right)$

Computer Implementation 5.3 *(Matlab)*

The analysis of two dimensional bounadry value problems using triangular elements can be performed conveniently by writing three *Matlab* functions, one for defining the element $\boldsymbol{k}_k + k_p$ and $\boldsymbol{r}_q$ vectors, one for evaluating natural boundary terms, and the third for computing the element solution.

MatlabFiles\Chap5\BVPTriElement.m

```
function [ke, rq] = BVPTriElement(kx, ky, p, q, coord)
% [ke, rq] = BVPTriElement(kx, ky, p, q, coord)
```

```
% Generates for a triangular element for 2d BVP
% kx, ky, p, q = parameters defining the BVP
% 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];
kp = -(p*A/12)*[2, 1, 1; 1, 2, 1; 1, 1, 2];
ke = kxx + kyy + kp;
rq = 1/3*q*A*[1; 1; 1];
```

MatlabFiles\Chap5\BVPTriNBCTerm.m

```
function [ka, rb] = BVPTriNBCTerm(side, alpha, beta, coord)
% [ka, rb] = BVPTriNBCTerm(side, alpha, beta, coord)
% Generates kalpha and rbeta when NBC is specified along a side
% side = side over which the NBC is specified
% alpha and beta = coefficients specifying the NBC
% 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);
   ka = -alpha*L/6 * [2,1,0; 1,2,0; 0,0,0];
   rb = beta *L/2 * [1; 1; 0];
case 2
   L = sqrt((x2-x3)^2+(y2-y3)^2);
   ka = -alpha*L/6 * [0,0,0; 0,2,1; 0,1,2];
   rb = beta *L/2 * [0; 1; 1];
case 3
   L = sqrt((x3-x1)^2+(y3-y1)^2);
   ka = -alpha*L/6 * [2,0,1; 0,0,0; 1,0,2];
   rb = beta *L/2 * [1; 0; 1];
end
```

MatlabFiles\Chap5\BVPTriResults.m

```
function results = BVPTriResults(coord, dn)
% results = BVPTriResults(coord, dn)
% Computes element solution for a triangular element for 2D BVP
% coord = nodal coordinates
% dn = nodal solution
% The results are computed at the element center
% The first three output variables are u and its x and y derivatives
% The last output variable is integral of solution over the element
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);
u = n*dn; dudx=[b1, b2, b3]*dn/(2*A); dudy=[c1, c2, c3]*dn/(2*A);
ui = sum(dn)*A/3;
results=[u, dudx, dudy, ui];
```

Using these functions now we consider solution of the torsion problem. The steps are exactly those used in other *Matlab* implementations.

MatlabFiles\Chap5\CShapeTorsionEx.m

```
% Torsion of a C Shaped Section
kx=1; ky=1; q=2; p=0;
nodes=[0,0; 0,6; 0.255,0;0.255,5.749499999999999;
   0.51,0; 0.51,5.499; 3.17,5.499; 3.17,5.749499999999999; 3.17,6];
lmm = [1,3,4; 4,2,1; 3,5,6; 6,4,3; 6,7,8; 8,4,6; 4,8,9; 9,2,4];
debc = [1,2,5,6,7,8,9]; ebcVals=zeros(length(debc),1);
dof=length(nodes); elems=size(lmm,1);
K=zeros(dof); R = zeros(dof,1);

% Generate equations for each element and assemble them.
for i=1:elems
   lm = lmm(i,:);
   [k, r] = BVPTriElement(kx, ky, p, q, nodes(lm,:));
   K(lm, lm) = K(lm, lm) + k;
   R(lm) = R(lm) + r;
end

% Nodal solution
```

```
d = NodalSoln(K, R, debc, ebcVals)
results=[];
for i=1:elems
   results = [results; BVPTriResults(nodes(lmm(i,:),:), d(lmm(i,:)))];
end
results
% Torsional constant calculations
J = 4*sum(results(:,4))
```

```
>> CShapeTorsionEx

d =

         0
         0
    0.0640
    0.0663
         0
         0
         0
         0
         0


results =

    0.0434    0.2511    0.0004    0.0318
    0.0221    0.2598         0    0.0169
    0.0213   -0.2511         0    0.0150
    0.0434   -0.2595    0.0004    0.0318
         0         0         0         0
    0.0221   -0.0227    0.2414    0.0081
    0.0221   -0.0227         0    0.0081
    0.0221         0   -0.2645    0.0088


J =

    0.4817
```

To get torsional constant we add all integrals over the elements (the fourth item in each element results). Multiplying this sum by 2 gives the total torque. Since we are modeling half of the C shape, the torque for the entire section is twice this value. Since $T = J\,G\,\theta$ and we have used $G = \theta = 1$, the computations show that the torsional constant J for the section is 0.482 in^4.