CHAPTER SIX

Mapped Elements

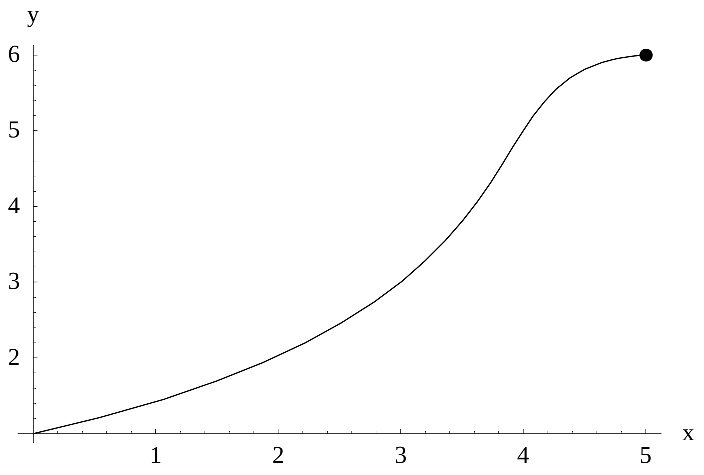
Computer Implementation 6.1 ***(Matlab)***

The calculations for mapping lines can be conveniently performed in *Mathematica*. Consider finding mapping for a line that passes through the following four points. The master element nodes are listed as well.

MatlabFiles\Chap6\LineMappingEx.m

```
% Mapping for a cubic line defined by the following four points
pts = [0,1; 3,3; 4,5; 5,6];
xn=pts(:,1); yn= pts(:,2);
map=[];
for s=-1:1/15:1
    % n = Cubic Lagrange interpolation functions
    n = [(-9*(-1 + s)*(-1/3 + s)*(1/3 + s))/16, ...
            (27*(-1 + s)*(-1/3 + s)*(1 + s))/16, ...
            (-27*(-1 + s)*(1/3 + s)*(1 + s))/16, ...
            (9*(-1/3 + s)*(1/3 + s)*(1 + s))/16];
    x = n*xn; y=n*yn;
    map=[map; [x,y]];
end
fprintf('Coordinates of point on the mapped line');
map'
plot(map(:,1), map(:,2))
```

```
>> LineMappingEx
Coordinates of point on the mapped line
ans =

  Columns 1 through 7

       0    0.4470    0.8560    1.2290    1.5680    1.8750    2.1520
    1.0000    1.1715    1.3520    1.5405    1.7360    1.9375    2.1440

  Columns 8 through 14

    2.4010    2.6240    2.8230    3.0000    3.1570    3.2960    3.4190
    2.3545    2.5680    2.7835    3.0000    3.2165    3.4320    3.6455

  Columns 15 through 21

    3.5280    3.6250    3.7120    3.7910    3.8640    3.9330    4.0000
    3.8560    4.0625    4.2640    4.4595    4.6480    4.8285    5.0000

  Columns 22 through 28

    4.0670    4.1360    4.2090    4.2880    4.3750    4.4720    4.5810
    5.1615    5.3120    5.4505    5.5760    5.6875    5.7840    5.8645

  Columns 29 through 31

    4.7040    4.8430    5.0000
    5.9280    5.9735    6.0000
```



Computer Implementation 6.2 *(Matlab)*

The calculations for mapping areas can easily be performed in *Matlab*. Consider finding mapping for a curved quadrilateral area. All four sides are curved and are defined by three points on each side for a total of 8 points. The master element has eight nodes as shown in the figure.

MatlabFiles\Chap6\TwoDMappingEx.m

```
% Mapping for a quadrilateral defined by the following eight points
pts = [0.5, 0.5; 1.1, 1.6; 1.7, 2.1; 1.5, 2.5; 1.1, 2.8;
   .75, 2.5; .5,2.5; .25, 1.5];
xn=pts(:,1); yn= pts(:,2);
map=[];
for s=-1:1/10:1
   for t=-1:1/10:1
      % 8 node serendipity interpolation functions
      n = [-((-1 + s)*(-1 + t)*(1 + s + t))/4, ...
            ((-1 + s^2)*(-1 + t))/2, ...
            ((-1 + t)*(1 - s^2 + t + s*t))/4,...
            -((1 + s)*(-1 + t^2))/2, ...
            ((1 + s)*(1 + t)*(-1 + s + t))/4, ...
            -((-1 + s^2)*(1 + t))/2, ...
            ((-1 + s)*(1 + s - t)*(1 + t))/4,...
            ((-1 + s)*(-1 + t^2))/2];
      dns = [-((-1 + t)*(2*s + t))/4, s*(-1 + t), ...
            ((-1 + t)*(-2*s + t))/4,...
            (1 - t^2)/2, ((1 + t)*(2*s + t))/4, ...
            -(s*(1 + t)), ((2*s - t)*(1 + t))/4, ...
            (-1 + t^2)/2];
```

```
        dnt = [((-1 + s)*(s + 2*t))/4, (-1 + s^2)/2, ...
              -((1 + s)*(s - 2*t))/4,...
              -((1 + s)*t), ((1 + s)*(s + 2*t))/4, ...
              (1 - s^2)/2, ...
              ((-1 + s)*(s - 2*t))/4, (-1 + s)*t];
        x = n*xn; y=n*yn;
        dxs=dns*xn; dxt=dnt*xn;
        dys=dns*yn; dyt=dnt*yn; J=[dxs, dxt; dys, dyt]; detJ = det(J);
        map=[map; [s, t, x, y, detJ]];
    end
end
detJ=[]; x=[]; y=[];
for i=1:21:length(map)
    x = [x, map(i:i+20,3)];
    y = [y, map(i:i+20,4)];
    detJ = [detJ, map(i:i+20,5)];
end
% Plot quadrilateral from mapped coordinates
% The surface plot shows detJ values
clf
hold on
mesh(x,y, zeros(length(x)));
mesh(x,y, detJ)
title('Jacobian over mapped element')
xlabel('x'); ylabel('y'); zlabel('detJ')
hold off
```

A plot of the actual element can be generated from the mapping by taking a series of points on the master element and generating x and y coordinates from the mapping. Comparing this plot to the actual element plot, it is visually clear that the mapping is valid.

Computer Implementation 6.3 *(Matlab)*

Evaluation of the following integral using five-point Gauss quadrature.

$$I = \int_{-1}^{1} (\mathrm{Exp}[s]\,\mathrm{Sin}[s]\,/\,(1+s^2))\,\mathrm{ds}$$

MatlabFiles\Chap6\OneDGaussQuadratureEx.m

```
% Integration using 5 point Gaussian quadrature
gaussPoints=[-0.906179845938664, -0.5384693101056831, 0.,...
    0.5384693101056831, 0.906179845938664];

gaussWeights=[0.2369268850561894, 0.47862867049936625, 0.5688888888888889,...
    0.47862867049936625, 0.2369268850561894];
```

```
int=0;
for i=1:length(gaussWeights)
   s = gaussPoints(i);
   fs = exp(s)*sin(s)/(1+s^2);
   int = int + gaussWeights(i)*fs;
end
int
```

```
>> OneDGaussQuadratureEx

int =

   0.4270
```

The built-in *Matlab* function quadl uses an adaptive approach. It keeps increasing the order of integration until the integral has been evaluated to a desired precision.

```
>> quadl('exp(s).*sin(s)./(1+s.^2)',-1,1)

ans =

   0.4274
```

Computer Implementation 6.4 *(Matlab)*

Consider evaluation of the following integral using Gauss quadrature.

$$I = \int_{-1}^{1}\int_{-1}^{1}(400\,s^5 + 675\,s^3 + 25\,s - 900\,s^2\,t^6 - 200\,t^2 + 0.2)\ \mathrm{ds}\ \mathrm{dt}$$

The built-in *Matlab* function dblquad can be used directly to integrate this function. The function uses an adaptive approach. It keeps increasing the order of integration until the integral has been evaluated to a desired precision.

```
>> dblquad(inline('0.2 + 25.*s - 200.*t.^2 + 675.*s.^3 - 900.*s.^2.*t.^6 + 400.*s.^5'), -1,1,-1,1)

ans =

 -437.2952
```

However, frequently in finite element computations, a more direct control is desired over the Gauss points used in the evaluation. In these situations we can take the product of locations and weights of the one dimensional Gauss quarature and then perform calculations as shown in the previous examples. For example to evaluate the given integral using 3×4 rule in *Matlab* these calculations can be conveniently organized as follows.

MatlabFiles\Chap6\TwoDGaussQuadratureEx.m

```
% Integration over a square using 4x4 Gaussian quadrature
gaussPoints=[-0.8611363115940526, -0.8611363115940526;
    -0.8611363115940526, -0.3399810435848563;
    -0.8611363115940526, 0.3399810435848563;
    -0.8611363115940526, 0.8611363115940526;
    -0.3399810435848563, -0.8611363115940526;
    -0.3399810435848563, -0.3399810435848563;
    -0.3399810435848563, 0.3399810435848563;
    -0.3399810435848563, 0.8611363115940526;
    0.3399810435848563, -0.8611363115940526;
    0.3399810435848563, -0.3399810435848563;
    0.3399810435848563, 0.3399810435848563;
    0.3399810435848563, 0.8611363115940526;
    0.8611363115940526, -0.8611363115940526;
    0.8611363115940526, -0.3399810435848563;
    0.8611363115940526, 0.3399810435848563;
    0.8611363115940526, 0.8611363115940526];

gaussWeights=[0.12100299328560216, 0.22685185185185194, ...
        0.22685185185185194, 0.12100299328560216, ...
        0.22685185185185194, 0.42529330301069407, ...
        0.42529330301069407, 0.22685185185185194, ...
        0.22685185185185194, 0.42529330301069407, ...
        0.42529330301069407, 0.22685185185185194, ...
        0.12100299328560216, 0.22685185185185194, ...
        0.22685185185185194, 0.12100299328560216];
int=0;
for i=1:length(gaussWeights)
    s = gaussPoints(i,1); t = gaussPoints(i,2);
    fst = 0.2 + 25*s + 675*s^3 + 400*s^5 - 200*t^2 - 900*s^2*t^6;
    int = int + gaussWeights(i)*fst;
end
int
```

```
>> TwoDGaussQuadratureEx

int =

 -437.2952
```

Comparing with the exact solution the 4×4 formula gives the exact integral. Thus any higher order integration formula will give the same answer.

Computer Implementation 6.5 *(Matlab)*

Consider evaluation of the following integral using Gauss quadrature.

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}(400\,t^5 + 675\,t^3 - 900\,s^4 - 200\,s^2 + 25\,r + 0.2)\,\mathrm{dr}\,\mathrm{ds}\,\mathrm{dt}$$

MatlabFiles\Chap6\ThreeDGaussQuadratureEx.m

```
% Integration over a cube using 1x2x3 Gaussian quadrature
gaussPoints=[0., -0.5773502691896257, -0.7745966692414834;
    0., -0.5773502691896257, 0.;
    0., -0.5773502691896257, 0.7745966692414834;
    0., 0.5773502691896257, -0.7745966692414834;
    0., 0.5773502691896257, 0.;
    0., 0.5773502691896257, 0.7745966692414834];

gaussWeights=[1.111111111111111, 1.777777777777777, ...
        1.111111111111111, 1.111111111111111, ...
        1.777777777777777, 1.111111111111111];
int=0;
for i=1:length(gaussWeights)
    r = gaussPoints(i,1); s = gaussPoints(i,2); t = gaussPoints(i,3);
    frst = 0.2 + 25*r - 200*s^2 + 675*t^3 - 900*s^4 + 400*t^5;
    int = int + gaussWeights(i)*frst;
end
int
```

```
>> ThreeDGaussQuadratureEx

int =

 -1.3317e+003
```

Computer Implementation 6.6 *(Matlab)* *4 Node Quadrilateral Element for 2D BVP*

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. The BVPQuad4Element employs 2×2 integration. The BVPQuad4NBCTerm employs two point integration. The functions can be modified easily to use any other integration formula. The BVPQuad4Results function computes results at the point of the element that corresponds to the origin of the master element ($s = t = 0$). The function returns the location of this point (in terms of x, y coordinates) and the solution and its x and y derivatives at this point. Obviously the function can be modified to compute results at any other point.

MatlabFiles\Chap6\BVPQuad4Element.m

```
function [ke, rq] = BVPQuad4Element(kx, ky, p, q, coord)
% [ke, rq] = BVPQuad4Element(kx, ky, p, q, coord)
% Generates for a 4 node quadrilateral element for 2d BVP
% kx, ky, p, q = parameters defining the BVP
% coord = coordinates at the element ends

% Use 2x2 integration. Gauss point locations and weights
pt=1/sqrt(3);
gpLocs = [-pt,-pt; -pt,pt; pt,-pt; pt,pt];
gpWts = [1,1,1,1];
kk=zeros(4); kp=zeros(4); rq=zeros(4,1);
for i=1:length(gpWts)
    s = gpLocs(i, 1); t = gpLocs(i, 2); w = gpWts(i);
    n = [(1/4)*(1 - s)*(1 - t), (1/4)*(s + 1)*(1 -t), ...
            (1/4)*(s + 1)*(t + 1), (1/4)*(1 - s)*(t + 1)];
    dns=[(-1 + t)/4, (1 - t)/4, (1 + t)/4, (-1 - t)/4];
    dnt=[(-1 + s)/4, (-1 - s)/4, (1 + s)/4, (1 - s)/4];
    x = n*coord(:,1); y = n*coord(:,2);
    dxs = dns*coord(:,1); dxt = dnt*coord(:,1);
    dys = dns*coord(:,2); dyt = dnt*coord(:,2);
    J = [dxs, dxt; dys, dyt]; detJ = det(J);
    bx = (J(2, 2)*dns - J(2, 1)*dnt)/detJ;
    by = (-J(1, 2)*dns + J(1, 1)*dnt)/detJ;
    b = [bx; by];
    c = [kx, 0; 0, ky];
    kk = kk + detJ*w* b'*c*b;
    kp = kp - detJ*w*p * n'*n;
    rq = rq + detJ*w*q * n';
end
ke=kk+kp;
```

MatlabFiles\Chap6\BVPQuad4NBCTerm.m

```
function [ka, rb] = BVPQuad4NBCTerm(side, alpha, beta, coord)
% [ka, rb] = BVPQuad4NBCTerm(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

% Use 2 point integration. Gauss point locations and weights
pt=-1/sqrt(3);
gpLocs = [-pt, pt];
gpWts = [1,1];
```

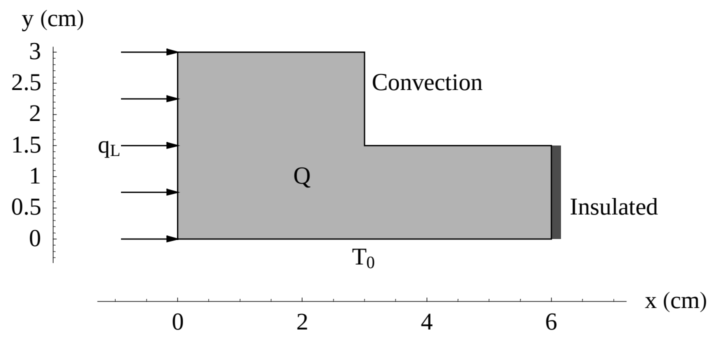
```
ka=zeros(4); rb=zeros(4,1);
for i=1:length(gpWts)
    a = gpLocs(i); w = gpWts(i);
    switch (side)
    case 1
        n = [(1 - a)/2, (1 + a)/2, 0, 0];
        dna = [-1/2, 1/2, 0, 0];
    case 2
        n = [0, (1 - a)/2, (1 + a)/2, 0];
        dna = [0, -1/2, 1/2, 0];
    case 3
        n = [0, 0, (1 - a)/2, (1 + a)/2];
        dna = [0, 0, -1/2, 1/2];
    case 4
        n = [(1 + a)/2, 0, 0, (1 - a)/2];
        dna = [1/2, 0, 0, -1/2];
    end
    dxa = dna*coord(:,1); dya = dna*coord(:,2);
    Jc=sqrt(dxa^2 + dya^2);
    ka = ka - alpha*Jc*w*n'*n;
    rb = rb + beta*Jc*w*n';
end
```

MatlabFiles\Chap6\BVPQuad4Results.m

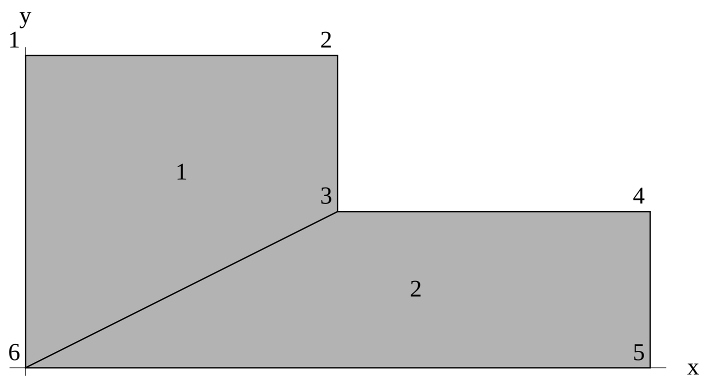
```
function results = BVPQuad4Results(coord, dn)
% results = BVPQuad4Results(coord, dn)
% Computes element solution for a quadrilateral element for 2D BVP
% coord = nodal coordinates
% dn = nodal solution
% The solution is computed at the element center
% The output variables are loc, u and its x and y derivatives
s = 0; t = 0;
n = [(1/4)*(1 - s)*(1 - t), (1/4)*(s + 1)*(1 -t), ...
        (1/4)*(s + 1)*(t + 1), (1/4)*(1 - s)*(t + 1)];
dns=[(-1 + t)/4, (1 - t)/4, (1 + t)/4, (-1 - t)/4];
dnt=[(-1 + s)/4, (-1 - s)/4, (1 + s)/4, (1 - s)/4];
x = n*coord(:,1); y = n*coord(:,2);
dxs = dns*coord(:,1); dxt = dnt*coord(:,1);
dys = dns*coord(:,2); dyt = dnt*coord(:,2);
J = [dxs, dxt; dys, dyt]; detJ = det(J);
bx = (J(2, 2)*dns - J(2, 1)*dnt)/detJ;
by = (-J(1, 2)*dns + J(1, 1)*dnt)/detJ;
b = [bx; by];
results=[x, y, n*dn, bx*dn, by*dn];
```

Heat flow in an L-shaped body using Quad4 elements

Consider two dimensional heat flow over an L-shaped body with thermal conductivity $k = 45\ W/m.°C$. The bottom is maintained at $T_0 = 110\ °C$. Convection heat loss takes place on the top where the ambient air temperature is $20\ °C$ and the convection heat transfer coefficient is $h = 55\ W/m^2.°C$. The right side is insulated. The left side is subjected to heat flux at a uniform rate of $q_L = 8000\ W/m^2$. Heat is generated in the body at a rate of $Q = 5\times 10^6\ W/m^3$. Determine temperature distribution in the body.



To demonstrate the use of the functions presented in this section we obtain a finite element solution of the problem using only 2 quadrilateral elements. Obviously we don't expect very good results. The mesh is chosen to simplify calculations. The steps are exactly those used in other *Matlab* implementations.



MatlabFiles\Chap6\LHeatQuad4Ex.m

```
% Heat flow through an L-shaped body using quad4 elements
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; 2, 2; 2, 1; 4, 1; 4, 0; 0, 0];
lmm = [6, 3, 2, 1; 6, 5, 4, 3];
debc = [5:6]; 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] = BVPQuad4Element(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] = BVPQuad4NBCTerm(4, 0, ql, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPQuad4NBCTerm(2, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPQuad4NBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(2,:);
[k, r] = BVPQuad4NBCTerm(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; BVPQuad4Results(nodes(lmm(i,:),:), ...
         d(lmm(i,:)))];
end
format short g
results
```

```
>> LHeatQuad4Ex

d =
  153.3936
  142.9067
  132.8533
  124.5394
  110.0000
  110.0000


results =

      0.015     0.01875       134.79       -90.82       1187.7
     0.0375      0.0075       119.35      -92.377       1338.8
```

Computer Implementation 6.7 *(Matlab)* *8 Node Quadrilateral Element for 2D BVP*

The functions for 8 node quadrilateral elements are almost identical to those for the four node quadrilater-las. The only changes are the dimensions of the matrices and that we use 3 points for integration and 8 node serendipity interpolation functions.

MatlabFiles\Chap6\BVPQuad8Element.m

```
function [ke, rq] = BVPQuad8Element(kx, ky, p, q, coord)
% [ke, rq] = BVPQuad8Element(kx, ky, p, q, coord)
% Generates for a 8 node quadrilateral element for 2d BVP
% kx, ky, p, q = parameters defining the BVP
% coord = coordinates at the element ends

% Use 3x3 integration. Gauss point locations and weights
gpWts = [5/9, 8/9, 5/9];
gpLocs = [-sqrt(3/5),-sqrt(3/5); 0,-sqrt(3/5); sqrt(3/5),-sqrt(3/5);
    -sqrt(3/5),0; 0,0; sqrt(3/5),0;
    -sqrt(3/5),sqrt(3/5); 0,sqrt(3/5); sqrt(3/5),sqrt(3/5)];
gpWts = [5/9 * 5/9, 8/9 * 5/9, 5/9 * 5/9, ...
        5/9 * 8/9, 8/9 * 8/9, 5/9 * 8/9, ...
        5/9 * 5/9, 8/9 * 5/9, 5/9 * 5/9];
kk=zeros(8); kp=zeros(8); rq=zeros(8,1);
for i=1:length(gpWts)
    s = gpLocs(i, 1); t = gpLocs(i, 2); w = gpWts(i);
    n = [((1 - s)*(1 - t))/4 - ((1 - s^2)*(1 - t))/4 - ...
            ((1 - s)*(1 - t^2))/4, ((1 - s^2)*(1 - t))/2, ...
            ((1 + s)*(1 - t))/4 - ((1 - s^2)*(1 - t))/4 - ...
            ((1 + s)*(1 - t^2))/4, ((1 + s)*(1 - t^2))/2, ...
            ((1 + s)*(1 + t))/4 - ((1 - s^2)*(1 + t))/4 - ...
            ((1 + s)*(1 - t^2))/4, ((1 - s^2)*(1 + t))/2, ...
            ((1 - s)*(1 + t))/4 - ((1 - s^2)*(1 + t))/4 - ...
        ((1 - s)*(1 - t^2))/4, ((1 - s)*(1 - t^2))/2];
    dns=[(s*(1 - t))/2 + (-1 + t)/4 + (1 - t^2)/4, -(s*(1 - t)), ...
        (1 - t)/4 + (s*(1 - t))/2 + (-1 + t^2)/4, (1 - t^2)/2, ...
        (1 + t)/4 + (s*(1 + t))/2 + (-1 + t^2)/4, -(s*(1 + t)), ...
        (-1 - t)/4 + (s*(1 + t))/2 + (1 - t^2)/4, (-1 + t^2)/2];
    dnt=[(-1 + s)/4 + (1 - s^2)/4 + ((1 - s)*t)/2, (-1 + s^2)/2, ...
        (-1 - s)/4 + (1 - s^2)/4 + ((1 + s)*t)/2, -((1 + s)*t), ...
        (1 + s)/4 + (-1 + s^2)/4 + ((1 + s)*t)/2, (1 - s^2)/2, ...
        (1 - s)/4 + (-1 + s^2)/4 + ((1 - s)*t)/2, -((1 - s)*t)];
    x = n*coord(:,1); y = n*coord(:,2);
    dxs = dns*coord(:,1); dxt = dnt*coord(:,1);
    dys = dns*coord(:,2); dyt = dnt*coord(:,2);
    J = [dxs, dxt; dys, dyt]; detJ = det(J);
    bx = (J(2, 2)*dns - J(2, 1)*dnt)/detJ;
    by = (-J(1, 2)*dns + J(1, 1)*dnt)/detJ;
    b = [bx; by];
    c = [kx, 0; 0, ky];
    kk = kk + detJ*w* b'*c*b;
    kp = kp - detJ*w*p * n'*n;
    rq = rq + detJ*w*q * n';
end
ke=kk+kp;
```

MatlabFiles\Chap6\BVPQuad8NBCTerm.m

```
function [ka, rb] = BVPQuad8NBCTerm(side, alpha, beta, coord)
% [ka, rb] = BVPQuad8NBCTerm(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

% Use 3 point integration. Gauss point locations and weights
gpLocs = [-sqrt(3/5), 0, sqrt(3/5)];
gpWts = [5/9, 8/9, 5/9];
ka=zeros(8); rb=zeros(8,1);
for i=1:length(gpWts)
    a = gpLocs(i); w = gpWts(i);
    switch (side)
    case 1
        n = [(1 - a)/2 + (-1 + a^2)/2, 1 - a^2, ...
            (1 + a)/2 + (-1 + a^2)/2, 0, 0, 0, 0, 0];
        dna = [-1/2 + a, -2*a, 1/2 + a, 0, 0, 0, 0, 0];
    case 2
        n = [0, 0, (1 - a)/2 + (-1 + a^2)/2, ...
            1 - a^2, (1 + a)/2 + (-1 + a^2)/2, 0, 0, 0];
        dna = [0, 0, -1/2 + a, -2*a, 1/2 + a, 0, 0, 0];
    case 3
        n = [0, 0, 0, 0, (1 - a)/2 + (-1 + a^2)/2, ...
                1 - a^2,(1 + a)/2 + (-1 + a^2)/2, 0];
        dna = [0, 0, 0, 0, -1/2 + a, -2*a, 1/2 + a, 0];
    case 4
        n = [(1 + a)/2 + (-1 + a^2)/2, 0, 0, 0, 0, 0, ...
                (1 - a)/2 + (-1 + a^2)/2, 1 - a^2];
        dna = [1/2 + a, 0, 0, 0, 0, 0, -1/2 + a, -2*a];
    end
    dxa = dna*coord(:,1); dya = dna*coord(:,2);
    Jc=sqrt(dxa^2 + dya^2);
    ka = ka - alpha*Jc*w*n'*n;
    rb = rb + beta*Jc*w*n';
end
```

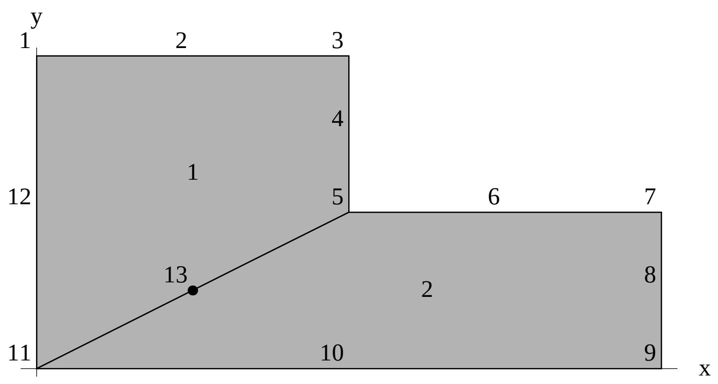
MatlabFiles\Chap6\BVPQuad8Results.m

```
function results = BVPQuad8Results(coord, dn)
% Computes element solution for a quadrilateral element for 2D BVP
% coord = nodal coordinates
% dn = nodal solution
% The solution is computed at the element center
% The output variables are loc, u and its x and y derivatives
s = 0; t = 0;
n = [((1 - s)*(1 - t))/4 - ((1 - s^2)*(1 - t))/4 - ...
        ((1 - s)*(1 - t^2))/4, ((1 - s^2)*(1 - t))/2, ...
        ((1 + s)*(1 - t))/4 - ((1 - s^2)*(1 - t))/4 - ...
        ((1 + s)*(1 - t^2))/4, ((1 + s)*(1 - t^2))/2, ...
        ((1 + s)*(1 + t))/4 - ((1 - s^2)*(1 + t))/4 - ...
        ((1 + s)*(1 - t^2))/4, ((1 - s^2)*(1 + t))/2, ...
        ((1 - s)*(1 + t))/4 - ((1 - s^2)*(1 + t))/4 - ...
        ((1 - s)*(1 - t^2))/4, ((1 - s)*(1 - t^2))/2];
dns=[(s*(1 - t))/2 + (-1 + t)/4 + (1 - t^2)/4, -(s*(1 - t)), ...
        (1 - t)/4 + (s*(1 - t))/2 + (-1 + t^2)/4, (1 - t^2)/2, ...
        (1 + t)/4 + (s*(1 + t))/2 + (-1 + t^2)/4, -(s*(1 + t)), ...
        (-1 - t)/4 + (s*(1 + t))/2 + (1 - t^2)/4, (-1 + t^2)/2];
dnt=[(-1 + s)/4 + (1 - s^2)/4 + ((1 - s)*t)/2, (-1 + s^2)/2, ...
        (-1 - s)/4 + (1 - s^2)/4 + ((1 + s)*t)/2, -((1 + s)*t), ...
        (1 + s)/4 + (-1 + s^2)/4 + ((1 + s)*t)/2, (1 - s^2)/2, ...
        (1 - s)/4 + (-1 + s^2)/4 + ((1 - s)*t)/2, -((1 - s)*t)];
x = n*coord(:,1); y = n*coord(:,2);
dxs = dns*coord(:,1); dxt = dnt*coord(:,1);
dys = dns*coord(:,2); dyt = dnt*coord(:,2);
J = [dxs, dxt; dys, dyt]; detJ = det(J);
bx = (J(2, 2)*dns - J(2, 1)*dnt)/detJ;
by = (-J(1, 2)*dns + J(1, 1)*dnt)/detJ;
```

```
b = [bx; by];
results=[x, y, n*dn, bx*dn, by*dn];
```

Heat flow in an L-shaped body using Quad8 elements

To demonstrate the use of the functions presented in this section we obtain a finite element solution of the problem using only 2 quadrilateral elements. Obviously we don't expect very good results. The mesh is chosen to simplify calculations. The steps are exactly those used in other *Mathematica* implementations.



MatlabFiles\Chap6\LHeatQuad8Ex.m

```
% Heat flow through an L-shaped body using quad8 elements
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; 2, 1.5; 2, 1; 3, 1;
    4, 1; 4, .5; 4, 0; 2, 0; 0, 0; 0, 1; 1, .5];
lmm = [11, 13, 5, 4, 3, 2, 1, 12;
    11, 10, 9, 8, 7, 6, 5, 13];
debc = [9:11]; 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] = BVPQuad8Element(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] = BVPQuad8NBCTerm(4, 0, ql, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPQuad8NBCTerm(2, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;
[k, r] = BVPQuad8NBCTerm(3, -h, htf, nodes(lm,:));
K(lm, lm) = K(lm, lm) + k;
R(lm) = R(lm) + r;

lm = lmm(2,:);
[k, r] = BVPQuad8NBCTerm(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; BVPQuad8Results(nodes(lmm(i,:),:), ...
            d(lmm(i,:)))];
end
format short g
results
```

```
>> LHeatQuad8Ex

d =

  156.4405
  150.7561
  149.1965
  144.2246
  133.8433
  124.0020
  121.7464
  119.1481
  110.0000
  110.0000
  110.0000
  144.6754
  129.1320


results =
```

0.015	0.01875	147.02	-255.3	961.07
0.0375	0.0075	122.24	-221.86	1155.3