Computer Implementation 6.7 ***(Matlab)*** *8 Node Quadrilateral Element for 2D BVP (p. 447)*

The functions for 8 node quadrilateral elements are almost identical to those for the four node quadrilaterlas. 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';
```

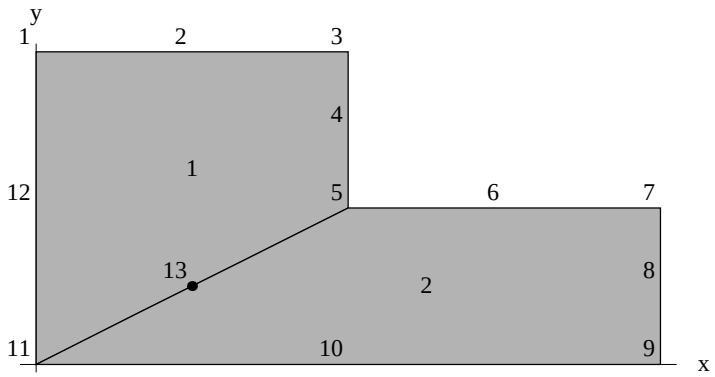
```
end
```

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
```