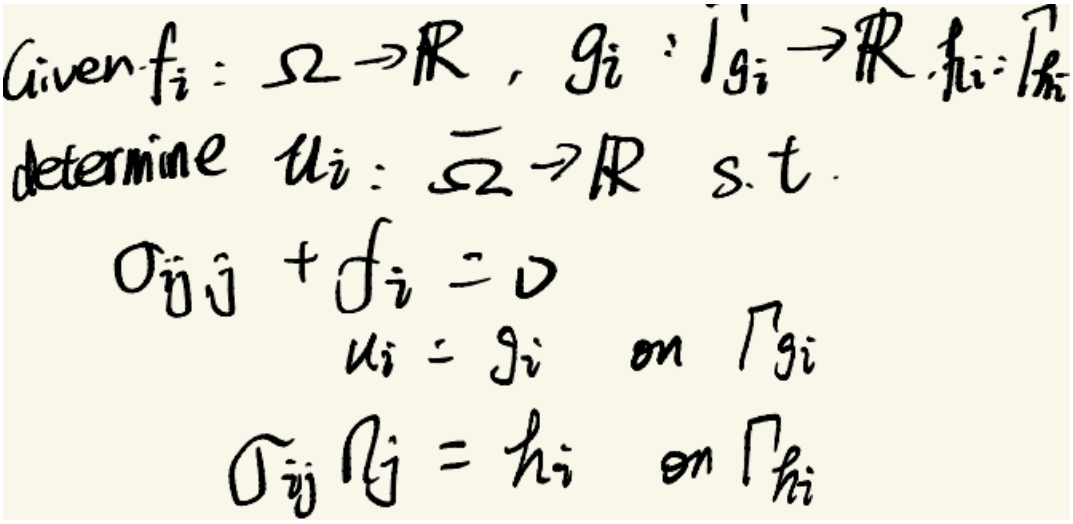
Computational Solid Mechanics Final Project Report

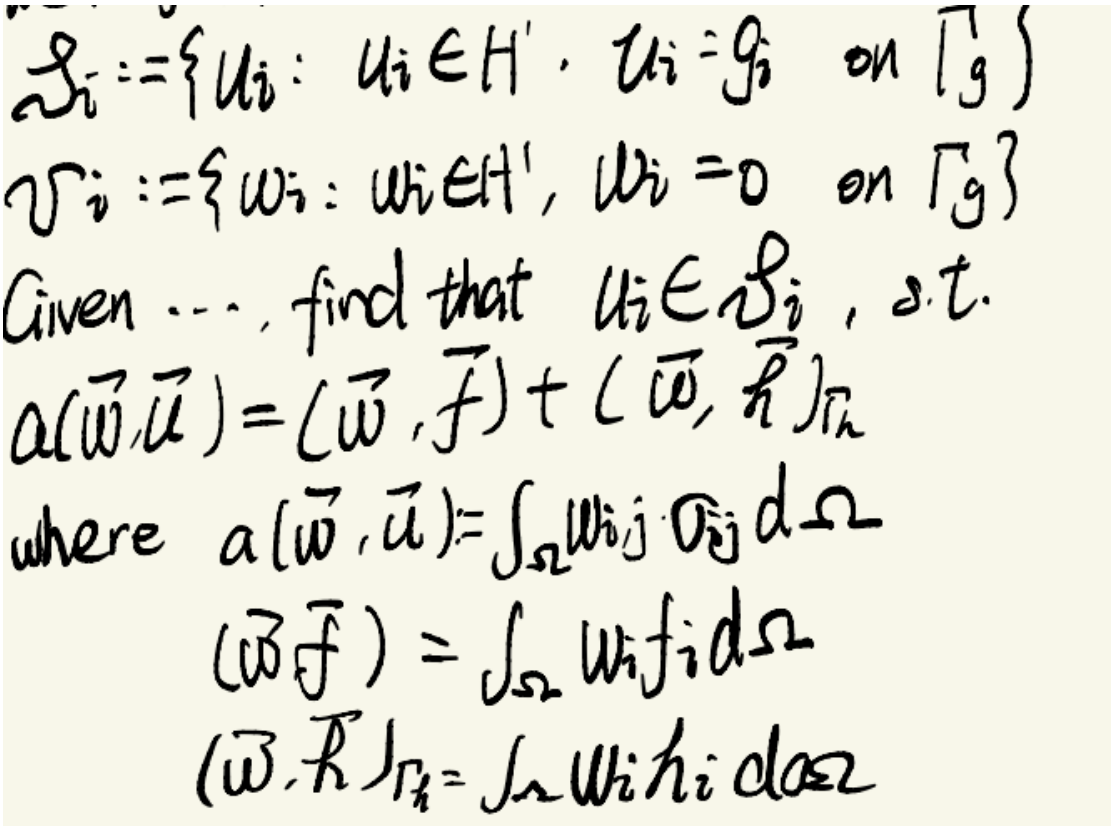
12110908 黄锦松

1. Problem Description

Strong form:



Weak form



Galerkin formulation

Galerkin formulation and implement

$$\mathcal{V}_i^h = \{ w_i^h : w_i^h(x_i) = \sum_{A \in \eta_B} C_{iA} N_A(x) \}$$

↑ i-th component

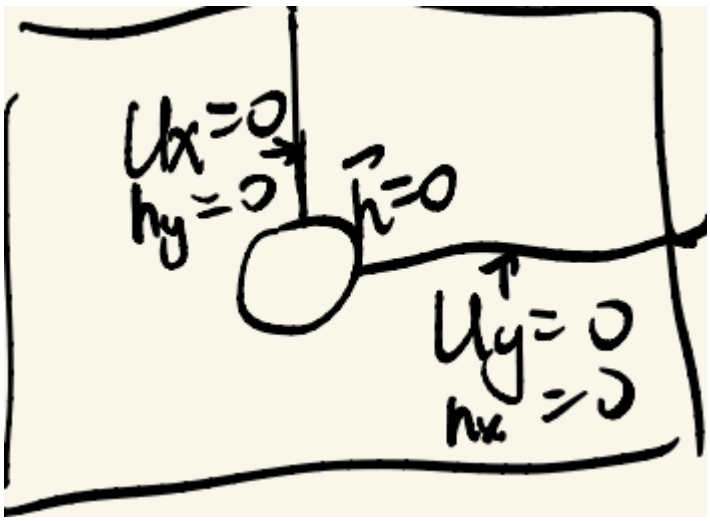
$$\mathcal{S}_i^h = \{ u_j^h = v_j^h + g_j^h,\ v_j^h = \sum_{B \in \eta_B} d_{jB} N_B,\ g_j^h = \sum_{B \in \eta_B} g_{iB} N_B \}$$

$$\sum_{A \in \eta_B} C_A \{ a(N_A \vec{e}_i, w^h) = (N_A \vec{e}_i, f) + (N_A \vec{e}_i, h)_\Gamma \}$$

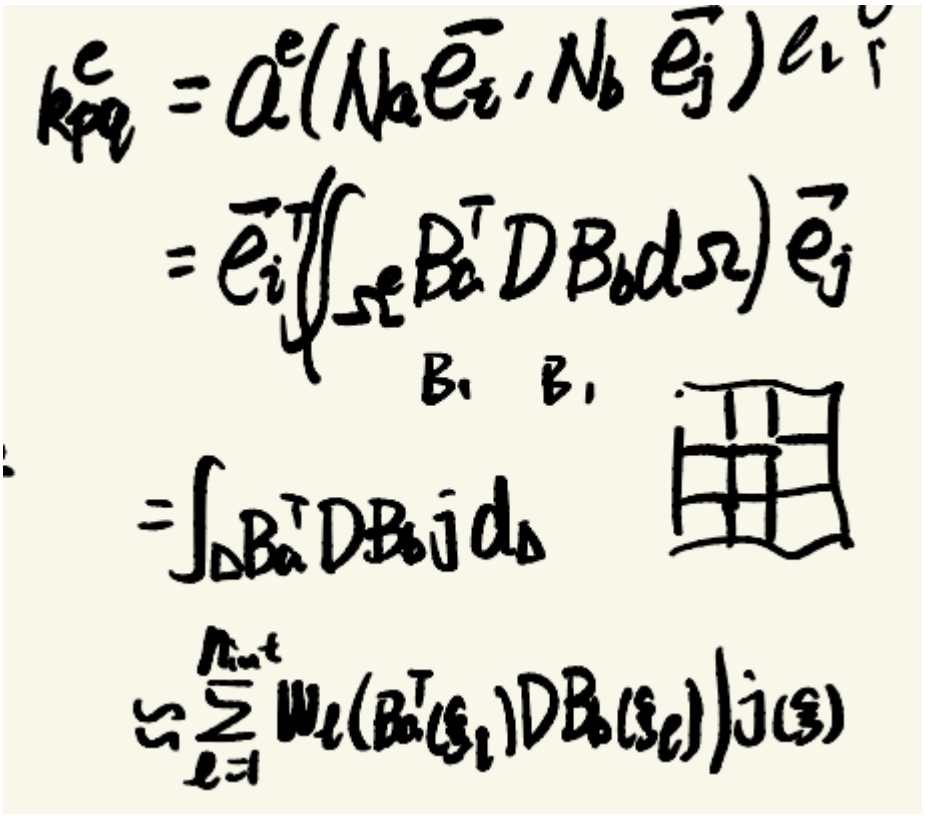
$$\Rightarrow a(N_A \vec{e}_i, N_B \vec{e}_j) d_{jB} = \int_\Omega N_A f_i \, d\Omega + \int_{\Gamma_h} N_A h_i \, da - a(N_A \vec{e}_i, \sum_{B \in \eta_B} g_{iB} N_B \vec{e}_j)$$

Boundary condition

For the outer surface, there are Dirichlet BC (e.g. g = 0) and Neumann BC (e.g. h = T). For inner hole, the BC is h = 0. For symmetry surface, the BC is like this



2. The implementation of the element stiffness matrix
   I choose $B_a^T D B_b$ implementation. Calculate B matrix first, then get k^e by matrix calculation. That is



3. Manufactured solution
   With given Tx = 10kpa, I calculate 3 stresses at each nodes as manufactured solution, then transfer the polar coordinates into Cartesian coordinates with equations below

$$\sigma_{rr}(r,\theta) = \frac{T_x}{2}\left(1-\frac{R^2}{r^2}\right)+\frac{T_x}{2}\left(1-4\frac{R^2}{r^2}+3\frac{R^4}{r^4}\right)cos2\theta,$$

$$\sigma_{\theta\theta}(r,\theta) = \frac{T_x}{2}\left(1+\frac{R^2}{r^2}\right)-\frac{T_x}{2}\left(1+3\frac{R^4}{r^4}\right)cos2\theta,$$

$$\sigma_{r\theta}(r,\theta) = -\frac{T_x}{2}\left(1+2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right)sin2\theta.$$

$$\sigma_x = \frac{\sigma_r+\sigma_\theta}{2}+\frac{\sigma_r-\sigma_\theta}{2}\cos 2\theta-\tau_{r\theta}\sin 2\theta$$

$$\sigma_y = \frac{\sigma_r+\sigma_\theta}{2}-\frac{\sigma_r-\sigma_\theta}{2}\cos 2\theta+\tau_{r\theta}\sin 2\theta$$

$$\tau_{xy} = \frac{\sigma_r-\sigma_\theta}{2}\sin 2\theta+\tau_{r\theta}\cos 2\theta$$

```
sigma_rr = @(Tx,R,r,theta) Tx./2.*(1 - R.^2./r.^2) + Tx./2.*(1 - 4.*R.^2./r.^2 + 3.*R.^4./r.^4).*cos(2.*theta);
sigma_tt = @(Tx,R,r,theta) Tx./2.*(1 + R.^2./r.^2) - Tx./2.*(1 + 3.*R.^4./r.^4).*cos(2.*theta);
sigma_rt = @(Tx,R,r,theta) -Tx./2.*(1 + 2.*R.^2./r.^2 - 3.*R.^4./r.^4).*sin(2.*theta);
sigma_rr = sigma_rr(Tx,R,r,theta);
sigma_tt = sigma_tt(Tx,R,r,theta);
sigma_rt = sigma_rt(Tx,R,r,theta);
% f
sigma_11 = (sigma_rr + sigma_tt)./2 + (sigma_rr - sigma_tt)./2.*cos(2.*theta) - sigma_rt.*sin(2.*theta);
sigma_22 = (sigma_rr + sigma_tt)./2 - (sigma_rr - sigma_tt)./2.*cos(2.*theta) + sigma_rt.*sin(2.*theta);
sigma_12 = (sigma_rr - sigma_tt)./2.*sin(2.*theta) + sigma_rt.*cos(2.*theta);
```

4. The codes are in driver.m. For the case I calculate, I define the right surface to be Dirichlet Boundary condition. I first load the .msh mesh data from gmsh and construct node coordinates and IEN array.

```
% 打开文件
fid = fopen('quarter-plate-with-hole-quad-Refine3.msh', 'r');

if fid == -1
    error('无法打开文件');
end

% 初始化
i = 1;
j = 1;
coor = zeros(1,3);
ien  = zeros(1,5);
% 读取文件内容
while ~feof(fid) ...
node_coor_1 = coor(2:563,:);
node_coor_1(:,3) = [];
node_coor_1(:,1) = node_coor_1(:,1) + 1;
node_coor_1(:,2) = node_coor_1(:,2) + 1;
IEN_1 = ien(8:519,2:5);
% 关闭文件
fclose(fid);
```

Then I calculate D matrix

```
E = 1e9; % Young's Modulus
pr = 0.3; % Poison's ratio
lambda = pr*E/(1+pr)/(1-2*pr);
mu = E/2/(1+pr);
% D = (E/(1-pr^2)).*[1,pr,0;pr,1,0;0,0,(1-pr)/2];
D = [lambda + 2*mu,lambda,0;lambda,lambda + 2*mu,0;0,0,mu];
```

Then construct ID and LM array. Notice that there is 2 degrees of freedom

```
% ID array
ID = zeros(n_np,2); % 2 dof
counter = 0;
for i = 1:n_np
    if i>67 && i<83
        counter = counter + 1;
        ID(i,1) = counter;
        counter = counter + 1;
        ID(i,2) = counter;
    else
        ID(i,1) = 0;
        ID(i,2) = 0;
    end
    if i>97
        counter = counter + 1;
        ID(i,1) = counter;
        counter = counter + 1;
        ID(i,2) = counter;
    end
end
n_eq = counter;

% LM array
ID1 = ID(:,1);
ID2 = ID(:,2);
LM1 = ID1(IEN_1);
LM2 = ID2(IEN_1);
LM = zeros(n_el,n_en*2);
for aa = 1 : n_en
    LM(:,2*aa-1) = LM1(:,aa);
    LM(:,2*aa) = LM2(:,aa);
end
```

Then I begin the numerical calculation part. There are 3 for loops. The biggest one is element loop. The second one is quadrature loop (I use quadrature rule provided in class for quadrilateral elements). The third one is element nodes loop. www

```
for ee = 1 : n_el
    x_ele = node_coor_1(IEN_1(ee, 1:n_en),1);
    y_ele = node_coor_1(IEN_1(ee, 1:n_en),2);

    k_ele = zeros(n_en*2, n_en*2); % element stiffness matrix
    f_ele_11 = zeros(n_en, 1);    % element load vector
    f_ele_22 = zeros(n_en, 1);
    f_ele = zeros(2*n_en,1);
```

Initiate element k matrix and f load vector.

```
for ll = 1 : n_int
    x_l = 0.0; y_l = 0.0;
    dx_dxi = 0.0; dx_deta = 0.0;
    dy_dxi = 0.0; dy_deta = 0.0;
    for aa = 1 : n_en
        x_l = x_l + x_ele(aa) * Quad(aa, xi(ll), eta(ll));
        y_l = y_l + y_ele(aa) * Quad(aa, xi(ll), eta(ll));
        [Na_xi, Na_eta] = Quad_grad(aa, xi(ll), eta(ll));
        dx_dxi  = dx_dxi  + x_ele(aa) * Na_xi;
        dx_deta = dx_deta + x_ele(aa) * Na_eta;
        dy_dxi  = dy_dxi  + y_ele(aa) * Na_xi;
        dy_deta = dy_deta + y_ele(aa) * Na_eta;
    end

    detJ = dx_dxi * dy_deta - dx_deta * dy_dxi;
```

Calculate Jacobian determinant.

```
for aa = 1 : n_en
    Na = Quad(aa, xi(ll), eta(ll));
    [Na_xi, Na_eta] = Quad_grad(aa, xi(ll), eta(ll));
    Na_x = (Na_xi * dy_deta - Na_eta * dy_dxi) / detJ;
    Na_y = (-Na_xi * dx_deta + Na_eta * dx_dxi) / detJ;

    f_ele_11(aa) = f_ele_11(aa) + weight(ll) * detJ * f_11(IEN_1(ee,aa)) * Na;
    f_ele_22(aa) = f_ele_22(aa) + weight(ll) * detJ * f_22(IEN_1(ee,aa)) * Na;
    f_ele(2*aa-1) = f_ele_11(aa);
    f_ele(2*aa) = f_ele_22(aa);

    Ba = zeros(3,2);
    Ba(1,2) = Na_x;
    Ba(2,2) = Na_y;
    Ba(3,1) = Na_y;
    Ba(3,2) = Na_x;
    for bb = 1:n_en
        Bb = zeros(3,2);
        Na = Quad(aa, xi(ll), eta(ll));
        [Na_xi, Na_eta] = Quad_grad(aa, xi(ll), eta(ll));
        Na_x = (Na_xi * dy_deta - Na_eta * dy_dxi) / detJ;
        Na_y = (-Na_xi * dx_deta + Na_eta * dx_dxi) / detJ;
        Bb(1,2) = Na_x;
        Bb(2,2) = Na_y;
        Bb(3,1) = Na_y;
        Bb(3,2) = Na_x;
        k_ij = Ba'*D*Bb;
        k_ele(2*aa-1:2*aa,2*bb-1:2*bb) = k_ele(2*aa-1:2*aa,2*bb-1:2*bb) + weight(ll).*detJ.*k_ij;
    end
end
```

Calculate element k and f using B’DB implementation. (Notice that there are two components for each element node)

```
for aa = 1 : n_en
    PP_1 = LM(ee, 2*aa-1);
    PP_2 = LM(ee, 2*aa);
    if PP_1 > 0
        F(PP_1) = F(PP_1) + f_ele(2*aa-1);

        for bb = 1 : n_en
            QQ_1 = LM(ee, 2*bb-1);
            QQ_2 = LM(ee, 2*bb);
            if QQ_1 > 0
                K(PP_1, QQ_1) = K(PP_1, QQ_1) + k_ele(2*aa-1, 2*bb-1);
            else
                % modify F with the boundary data
                % here we do nothing because the boundary data g is zero or
                % homogeneous
            end
            if QQ_2 > 0
                K(PP_1, QQ_2) = K(PP_1, QQ_2) + k_ele(2*aa-1, 2*bb);
            else
                % modify F with the boundary data
                % here we do nothing because the boundary data g is zero or
                % homogeneous
            end
        end
    end
    if PP_2 > 0
        F(PP_2) = F(PP_2) + f_ele(2*aa);

        for bb = 1 : n_en
            QQ_1 = LM(ee, 2*bb-1);
            QQ_2 = LM(ee, 2*bb);
            if QQ_1 > 0
                K(PP_2, QQ_1) = K(PP_2, QQ_1) + k_ele(2*aa, 2*bb-1);
            else
                % modify F with the boundary data
                % here we do nothing because the boundary data g is zero or
                % homogeneous
            end
            if QQ_2 > 0
                K(PP_2, QQ_2) = K(PP_1, QQ_2) + k_ele(2*aa, 2*bb);
            else
                % modify F with the boundary data
                % here we do nothing because the boundary data g is zero or
                % homogeneous
            end
        end
    end
end
```

Assembly of big K matrix and F load vector. (Still, there are two components for each node. So compared to heat conduction problem, the size of K and F is twice as large.

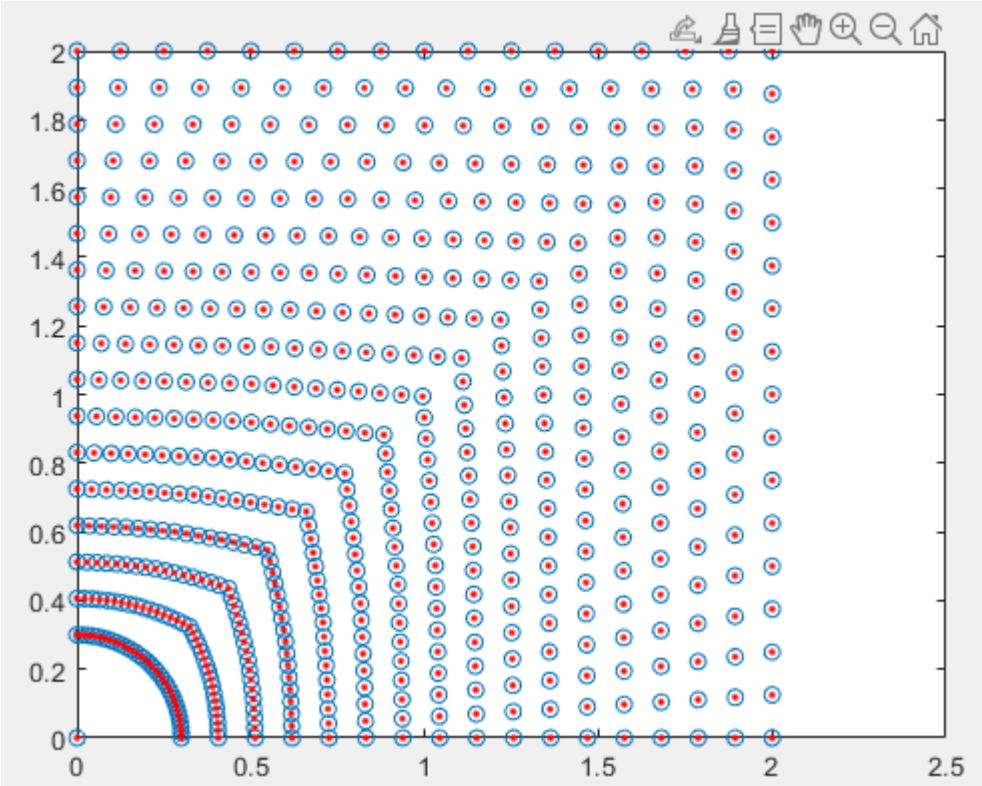
```
d = K\F;
d_x = zeros(length(d)/2,1);
d_y = zeros(length(d)/2,1);
distance = zeros(length(d)/2,1);
for i = 1:length(d)/2
    d_x(i) = d(2*i-1);
    d_y(i) = d(2*i);
    distance(i) = sqrt(d_x(i)^2+d_y(i)^2);
end
disp = zeros(n_np, 2);

for ii = 1 : n_np
  index_x = ID(ii,1);
  index_y = ID(ii,2);
  if index_x > 0
    disp(ii,1) = d(index_x);
  else
    % modify disp with the g data. Here it does nothing because g is zero
  end
  if index_y > 0
      disp(ii,2) = d(index_y);
  else
      % modify disp with the g data. Here it does nothing because g is zero
  end
end
```

```
%%

plot(node_coor_1(:,1),node_coor_1(:,2),'o')
hold on
plot(node_coor_1(:,1)+disp(:,1),node_coor_1(:,2)+disp(:,2),'.r')
```

Then I solve the matrix and reconstruct the displacement vector with given boundary data. Then visualization



(Blue circle is initial nodes, red dots is the nodes after displacement)
The difference is too small to be observed