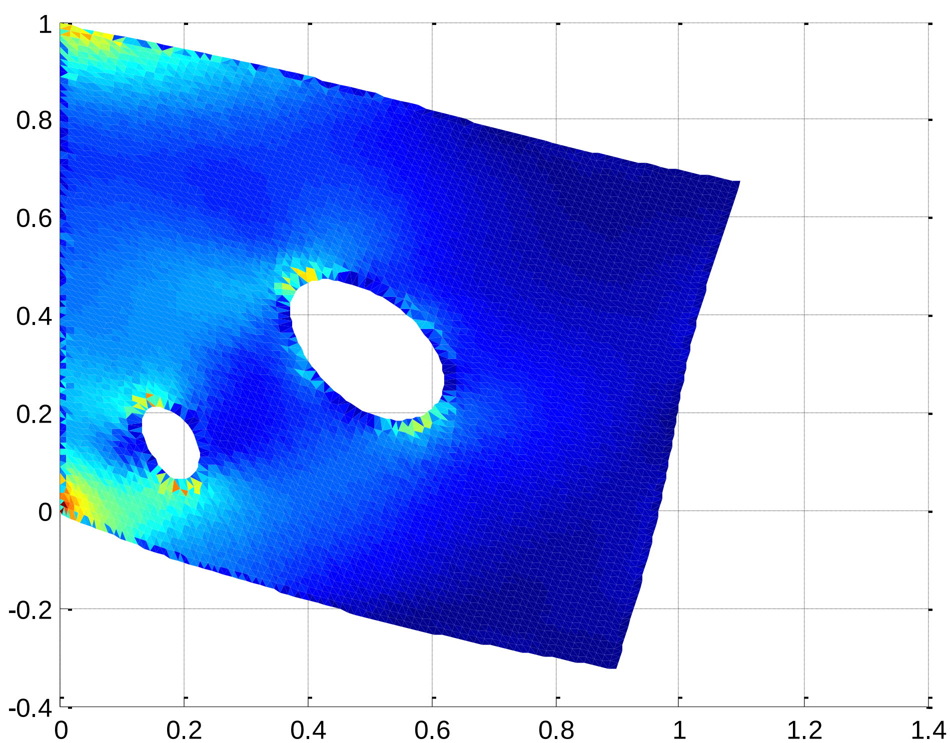
Problem 3

Program Log:

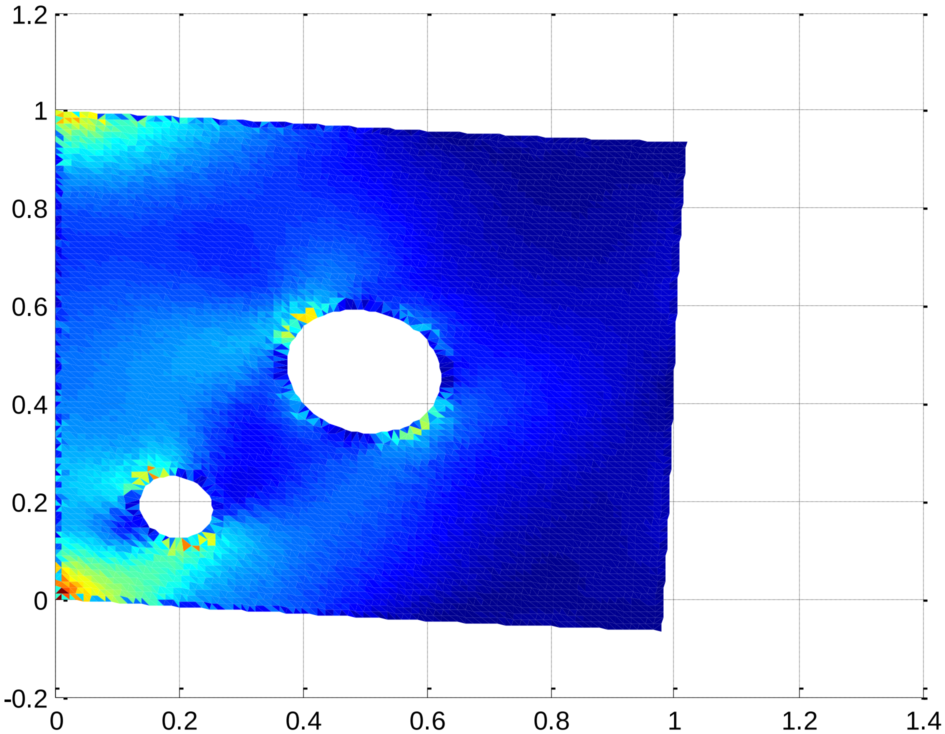
Linear elasticity equilibrium 2D FEM Solver with zero Dirichlet boundary condition
Time cost for loading mesh:0.10109
Time cost for computing boundary segment mesh:0.92969
Time cost for identifying dirichlet nodes:0.022701
Time cost for building linear system:18.1015
minres stopped at iteration 1000 without converging to the desired tolerance 1e-010
because the maximum number of iterations was reached.
The iterate returned (number 1000) has relative residual 5e-008.
Time cost for solving linear system:1.8682
Time cost for evaluating elementwise Cauchy stress:0.87304
Time cost for plotting:3.5037
END

Solution Plot:

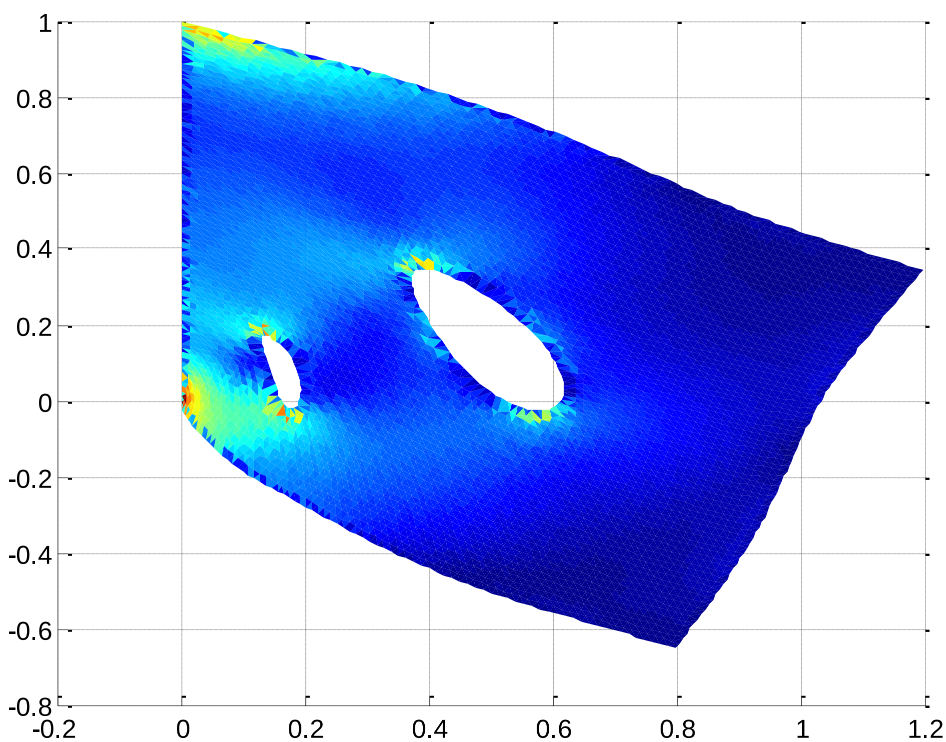
youngs_modulus = **100**;
poisson_ratio = 0.3;
density = 1**;**



youngs_modulus = **500**;
poisson_ratio = 0.3;
density = 1;



youngs_modulus = **50**;
poisson_ratio = 0.3;
density = 1;



MATLAB CODE

Most mesh and FEM functions are the same with Poisson.

Additional Functions are:

1. Indentify_dirichlet_nodes
2. Evaluate_cauchy_stres

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Linear elasticity equilibrium 2d FEM solver
%
% Solves div(sigma) + f = 0
%       u = 0 at dirichelt boundary
% over a triangle mesh
%
% TODO: let it support arbitrary dirichlet value (currently only support 0)
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; close all; clc
fprintf('###  Linear elasticity equilibrium 2D FEM Solver with zero Dirichlet boundary condition  ###\n')

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% input the problem
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

youngs_modulus = 100;
poisson_ratio = 0.3;
density = 1;

dirichlet_box = [1e-8, 999, -999, 999]; % this is a [xmin xmax ymin ymax] box that cuts out dirichlet nodes.
dirichlet_value = 0; % currently this program only supports 0.

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% material and world parameters
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

lambda = youngs_modulus*poisson_ratio / ((1 + poisson_ratio)*(1 - 2*poisson_ratio));
mu = youngs_modulus / ( 2 * (1 + poisson_ratio) );
gravity = density * [0, -9.8]';

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% some pre-processing for the mesh
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

tic;

% load mesh from file
elements = load('mesh_with_holes.dat');                       % element data
N_elements = size(elements,1);
nodes = load('nodes.dat');                                    % node data
N_nodes = size(nodes,1);
time_cost = toc; display(strcat('Time cost for loading mesh: ',num2str(time_cost))); tic;
```

```
% compute boundary segment mesh
boundary_segments = generate_boundary_segments_from_mesh(elements,nodes);    % boundary segment data
boundary_nodes = boundary_segments(:,1);                                     % boundary node data
time_cost = toc; display(strcat('Time cost for computing boundary segment mesh: ',num2str(time_cost))); tic;

% identify dirichlet boundary
[dirichlet_data dirichlet_node_list] = identify_dirichlet_nodes(nodes, dirichlet_box(1), dirichlet_box(2),
dirichlet_box(3), dirichlet_box(4), dirichlet_value);
time_cost = toc; display(strcat('Time cost for identifying dirichlet nodes: ',num2str(time_cost))); tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build and solve FEM system
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[K rhs] = build_system(elements,nodes,dirichlet_data,dirichlet_node_list,lambda,mu,gravity);
time_cost = toc; display(strcat('Time cost for building linear system: ',num2str(time_cost))); tic;

u = minres(K,rhs,1e-10,1000);
time_cost = toc; display(strcat('Time cost for solving linear system: ',num2str(time_cost))); tic;

stress = evaluate_stress(elements,nodes,u,lambda,mu);
time_cost = toc; display(strcat('Time cost for evaluating elementwise Cauchy stress: ',num2str(time_cost))); tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% plot the final solution u
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% compute the node stress
node_stress=zeros(N_nodes,1);
for t=1:N_elements
    s_norm=sqrt(stress(t,1)*stress(t,1)+stress(t,2)*stress(t,2)+stress(t,3)*stress(t,3)+stress(t,4)*stress(t,4));
    for i=1:3
        node_stress(elements(t,i))=node_stress(elements(t,i))+s_norm/3;
    end
end

% compute world space coordinates
for i=1:N_nodes
    x(i)=nodes(i,1);
    y(i)=nodes(i,2);
    x_deformed(i)=x(i)+u(2*i-1);
    y_deformed(i)=y(i)+u(2*i);
end

% plot the material space
figure
triplot(elements,x,y);
title('material space mesh')

% plot the world space with stress color
figure
trisurf(elements,x_deformed,y_deformed,node_stress,'EdgeColor','none');
```

```
view(2)

time_cost = toc; display(strcat('Time cost for plotting: ',num2str(time_cost))); tic;
fprintf('END\n')



function [dirichlet_data dirichlet_node_list]= identify_dirichlet_nodes(nodes, xmin, xmax, ymin, ymax,
dirichlet_value)
% dirichlet nodes are nodes cutted out from the input box
% dirichlet_data:
% the first column is bool, flag of whether a node is dirichlet
% the second column is the corresponding dirichlet value

N = size(nodes,1);

dirichlet_data = zeros(N,2);
dirichlet_node_list = [];

for i = 1:N
    x = nodes(i,1);
    y = nodes(i,2);

    if x<xmin || x>xmax || y<ymin || y>ymax
        dirichlet_data(i,1) = 1;
        dirichlet_data(i,2) = dirichlet_value;

        dirichlet_node_list = [dirichlet_node_list, i];
    end
end

end

function stress = evaluate_stress(elements,nodes,u,lambda,mu)

N_elements = size(elements,1);
N_nodes = size(nodes,1);

stress = zeros(N_elements,4);

for e = 1:N_elements
    element = elements(e,:);
    X1 = nodes(element(1),:);
    X2 = nodes(element(2),:);
    X3 = nodes(element(3),:);

    area = compute_element_area(elements,nodes,e);

    Dm = [(X2-X1)' (X3-X1)'];
    Dm_inv = inv(Dm);
    a = Dm_inv(1,1);
    b = Dm_inv(1,2);
```

```
    c = Dm_inv(2,1);
    d = Dm_inv(2,2);

    u1 = [u(2*element(1)-1) u(2*element(1))];
    u2 = [u(2*element(2)-1) u(2*element(2))];
    u3 = [u(2*element(3)-1) u(2*element(3))];

    Du = [(u2-u1)' (u3-u1)'];

    du_dx = Du*Dm_inv;
    strain = 0.5*(du_dx+du_dx');
    sigma = 2*mu*strain + lambda*trace(strain);

    stress(e,:) = [sigma(1,1), sigma(1,2), sigma(2,1), sigma(2,2)];

end

function [K rhs] = build_system(elements,nodes,dirichlet_data,dirichlet_node_list,lambda,mu,gravity)

N_elements = size(elements,1);
N_nodes = size(nodes,1);

K = sparse(2*N_nodes,2*N_nodes);
rhs = zeros(2*N_nodes,1);

% build matrix
for e = 1:N_elements
    element = elements(e,:);
    X1 = nodes(element(1),:);
    X2 = nodes(element(2),:);
    X3 = nodes(element(3),:);

    area = compute_element_area(elements,nodes,e);

    Dm = [(X2-X1)' (X3-X1)'];
    Dm_inv = inv(Dm);
    a = Dm_inv(1,1);
    b = Dm_inv(1,2);
    c = Dm_inv(2,1);
    d = Dm_inv(2,2);

    M = [-(a+c)   0   a    0   c    0;
         0     -(a+c) 0    a   0    c;
         -(b+d)   0   b    0   d    0;
         0     -(b+d) 0    b   0    d];

    M_hat = [1   0    0    0;
             0  0.5  0.5   0;
             0  0.5  0.5   0;
             0   0    0    1] * M;
    Ke = M_hat' * [2*mu+lambda  0    0      lambda ;
                   0       2*mu  0          0  ;
```

```
                 0        0   2*mu        0  ;
                lambda     0    0     2*mu+lambda] * M_hat;

    for ie = 1:3
        i = elements(e,ie);
        for je = 1:3
            j = elements(e,je);
            for a = 1:2
                for b = 1:2
                    K(2*(i-1)+a, 2*(j-1)+b) = K(2*(i-1)+a, 2*(j-1)+b) + area*Ke(2*(ie-1)+a, 2*(je-1)+b);
                end
            end
        end
    end
end

% build rhs
for e = 1:N_elements

    area = compute_element_area(elements,nodes,e);

    Fe = (1/3)*area*[gravity;gravity;gravity];

    for ie = 1:3
        i = elements(e,ie);
        rhs(2*(i-1)+2) = rhs(2*(i-1)+2) + Fe(2*(ie-1)+2);
    end
end

% impose zero dirichlet boundary
for it = 1:size(dirichlet_node_list,2)
    i = dirichlet_node_list(it);
    K(2*(i-1)+1,:) = 0;
    K(:,2*(i-1)+1) = 0;
    K(2*(i-1)+1,2*(i-1)+1) = 1;
    rhs(2*(i-1)+1) = 0;

    K(2*(i-1)+2,:) = 0;
    K(:,2*(i-1)+2) = 0;
    K(2*(i-1)+2, 2*(i-1)+2) = 1;
    rhs(2*(i-1)+2) = 0;
end


end % end of function: build_system
```