Project Assignment #2
28 November 2018

MAE 610 Numerical Methods for Engineers
Dr. Loren Sumner

Jordan Hughes and Hamilton Wise

Abstract

This project models the cross-sectional flow field in a conduit defined by fluid passing between a rectangular outer shell and a circular internal pipe along the center axis. First, the problem will be given mathematically showing the governing equation with boundary conditions. Then, using the finite element method, the problem will be solved in Matlab through use of the Galerkin method for the numerical approximation. The model is displayed using a contour plot to show the fluid velocity at different coordinates within the conduit.

Mathematical Problem

In the study if fluid mechanics, laminar flow in a conduit can be modeled as being unidirectional. The following scaled mathematical problem governs the fluid velocity, u, in the cross-section (x, y) of the conduit.

$$\nabla^2 u = f(x, y) = -1 \tag{1}$$

The conduit shall be defined as the area between a 8x10-rectangular outer shell and an internal pipe along the center axis with a diameter of 4. The boundary conditions state that the fluid velocity is 0 on the conduit walls, written as u = 0. Figure 1 illustrates the system with the governing equation and boundary conditions on the conduit.

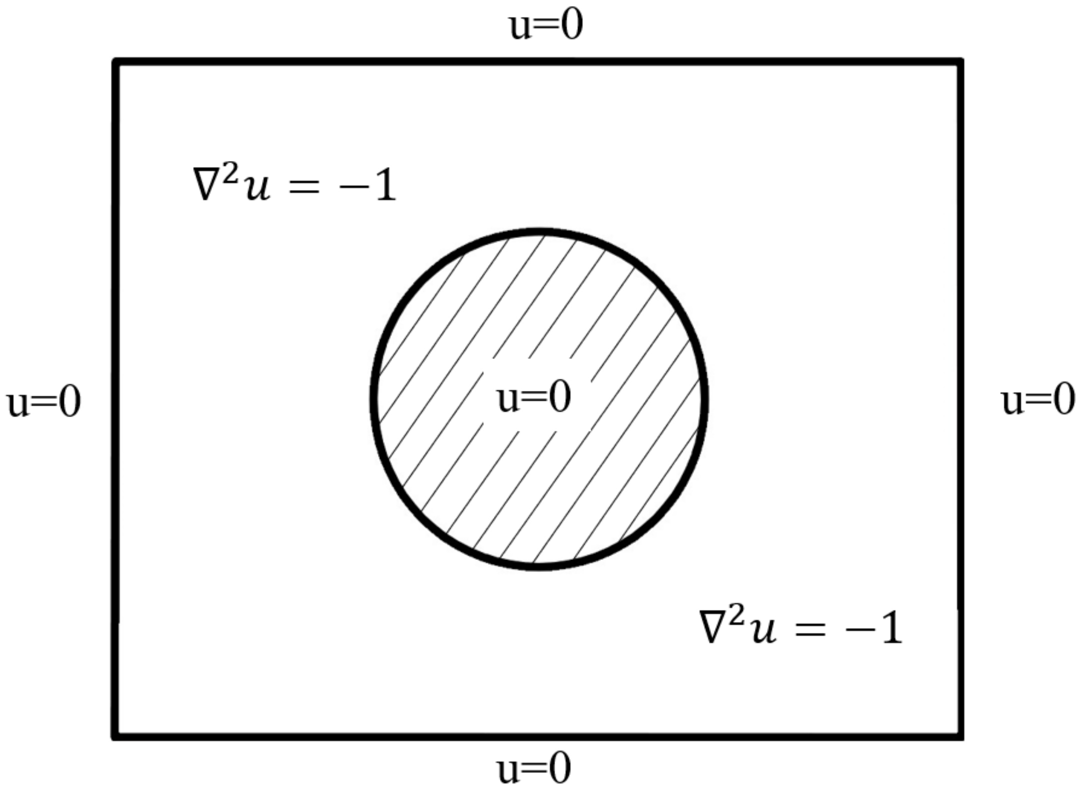


Figure 1: *Conduit Shape with Boundary Conditions*

The total flow rate through the conduit can be calculated by taking a double integral over the area of the region, as shown below.

$$Q = \iint u dA \tag{2}$$

Applied FE Theory

The Finite Element Method is a numerical way to approximate partial differential equations that are often too complex to solve analytically. The first step of the finite element method is to create a mesh over the physical model that is being analyzed. The mesh is composed of a number of subdomains known as elements which are defined by nodes at the vertex of each element. The element and node information are stored in connectivity matrices which are used when numerically solving the mathematical model.

In a 2D domain, every node has an X and Y coordinate that defines the location of that particular node. The mesh connectivity matrix, defined as “XY” in the code, stores the X and Y coordinates for every node in the model. The nodal connectivity matrix defines the mesh by storing each node number for each element and is defined in the code as “CONN”. These matrices are used to create the shape functions that are used when creating a local stiffness matrix and load vector for each element. The local matrices for each element are combined to create a global stiffness matrix and load vector which will then be solved to approximate the governing differential equation. The equation below shows how the stiffness matrix, load vector, and unknown nodal vector are related.

$$[k]\{U\} = \{b\} \tag{3}$$

The size of the global stiffness matrix and load vector correspond to the number of nodes that define the system. After the global stiffness matrix and load vector are created, the known boundary conditions are applied and then used to update the global stiffness matrix and load vector. The boundary condition for each node is used to update in the global load vector with the known value. The corresponding row for each updated node in the global stiffness matrix is then replaced with all zeros and the column in the row for the boundary node is replaced with a one.

The global stiffness matrix and load vector creates a system of linear equations, and is then numerically solved to find the unknown nodal values. The nodal values are then interpolated over each element and this is the numerical approximation for the mathematical model that governs the system.

Numerical Procedures and Programming

Using the Galerkin method, consider Poisson's equation given in equation 1 and let

$$u = \sum_j N_j U_j \tag{4}$$

Next, get the residual of equation 1 by solving for 0.

$$\mathrm{R(x,y)} = \nabla^2 u - f(x,y) = 0 \tag{5}$$

Then, setting the weighted total residual to zero by N_i and combining with equation 2 results in equation 6.

$$\iint N_i[\nabla^2 u - f(x,y)]dA = 0 \tag{6}$$

By reducing the order of ∇^2u and using Green's theorem, equation 6 becomes equation 7.

$$-\iint N_{ix}u_x + N_{iy}u_y dA + \iint f(x,y)N_i dA + \oint N_i \tag{7}$$

Recalling that f(x,y) = -1 given in the problem and combining with equation 4, equation 7 can be rewritten as equation 8.

$$\sum_j \iint N_{ix}N_{jx} + N_{iy}N_{jy} dA\, U_j = \iint N_i dA + \oint N_i \frac{\partial u}{\partial n} dS \tag{8}$$

Now, the term $\iint N_i dA$ allows for the application of the function (f(x,y) = −1) to the elements and $\oint N_i(\partial u/\partial n)dS$ allows for the cancellation of internal element boundary terms. For a 3-node element the fluid velocity u would be equation 9.

$$u = N_1(x,y)U_1 + N_2(x,y)U_2 + N_3(x,y)U_3 \tag{9}$$

In matrix form, combining equation 8 with equation 9 becomes

$$[A]\begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} \iint N_1 dA + \oint N_1 \frac{\partial u}{\partial n} dS \\ \iint N_2 dA + \oint N_2 \frac{\partial u}{\partial n} dS \\ \iint N_3 dA + \oint N_3 \frac{\partial u}{\partial n} dS \end{Bmatrix} \tag{10}$$

From equation 3, the load vector $\{b\}$ is the right matrix including the integrals, the unknown nodal value matrix $\{U\}$ is U_1, U_2, and U_3, and the stiffness matrix is [A]. For the special case of a 3-node triangular element, the stiffness matrix [A] can be assembled using equation 11,

$$A_{ij} = \iint N_{ix}N_{jx} + N_{iy}N_{jy} dA = (b_i b_j + c_i c_j)A_e \tag{11}$$

where $A_e$is the area of the element. In equation 11, the constants *b* and *c* are calculated at each node using equations 12 and 13,

$$b_k = \frac{-1}{2A_e}(y_l - y_m) \tag{12}$$

$$c_k = \frac{1}{2A_e}(x_l - x_m) \tag{13}$$

where $b_k$and c_k are calculated for each node in the element and *l* and *m* represent the adjacent nodes of the element.

The detailed mesh for this model is shown in Figure 2 and was generated using the *plotMesh* function in line 169 of the code shown in the appendix.

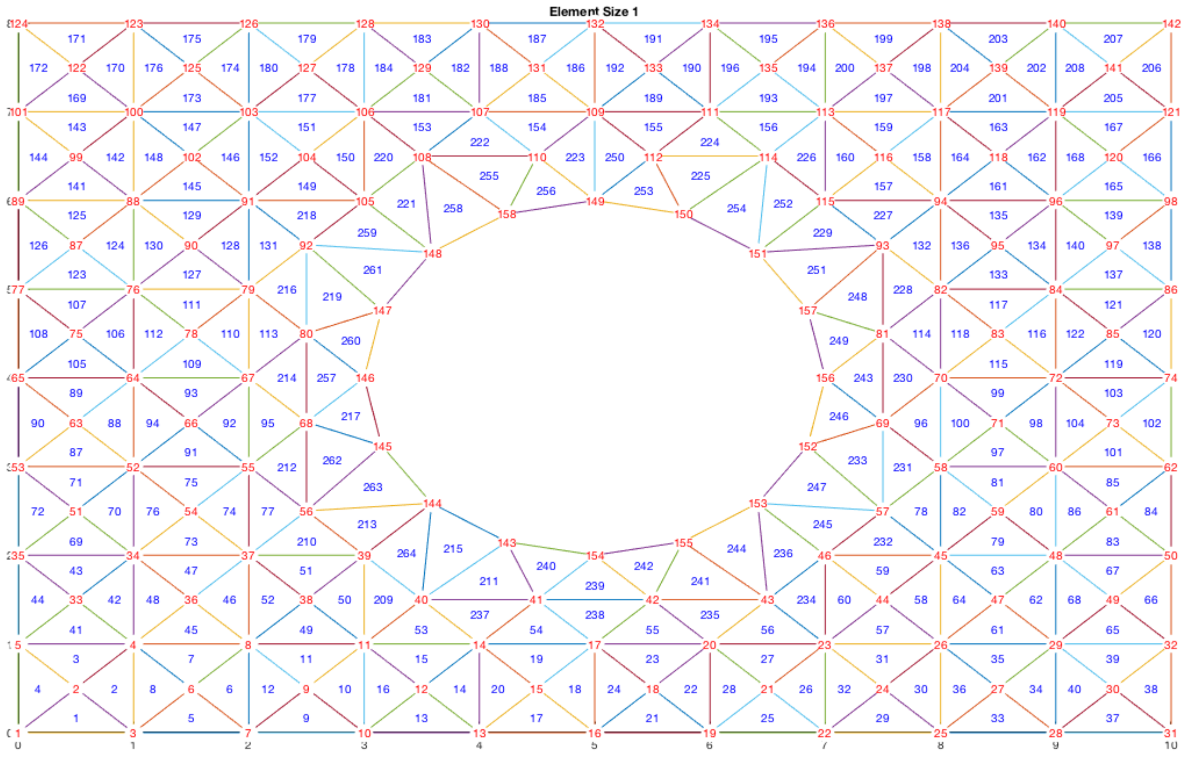


***Figure 2**: Mesh for Element Length 1*

The program steps through each element of the mesh creating a local stiffness matrix and load vector that applies to that specific element. The node numbers that define each element are stored in the CONN matrix. Each element in the mesh is defined by the node numbers stored in the corresponding row number of the CONN matrix. Each number from the CONN matrix correspond to the row in the XY matrix that stores each node's position in the physical model.

To assemble the local stiffness matrix and load vector, the x, y-coordinates that correspond to every node defining a specific element are used in 2D Simplex Elements formulas as shown in equations 12 and 13 and were evaluated using the *simplexElements* function shown in line 150 of the attached code. Evaluating equation 11 with the values found using equations 12 and 13 will construct the local stiffness matrix for that element. The load vector for the same element is found using Equation 10 which for the case of a 2-simplex element is equal to ⅓ of the area. The local stiffness matrix and load vector was generated for each element as shown in lines 35 and 37 of the attached code.

After the local stiffness matrix and load vector is created, those values are then placed into the global stiffness and global load vector in the row and column of the node numbers used to create the local stiffness and load matrix. Lines 42 and 44 show the assembly of the global stiffness matrix and global load vector. After every element's local stiffness matrix and load vector has been assembled into the global stiffness and load matrix, the boundary conditions must be applied to make the system of linear equations non-singular.

For this problem, the boundary condition states that velocity is zero on the conduit walls. For every node on the conduit wall, a zero was updated into the load vector corresponding to that node number. The row of the stiffness matrix was updated with all zeros except for a one which was placed in the column corresponding to each boundary node number. Once the boundary conditions have been applied to the linear system, the linear system can then be solved resulting in the velocity at every node defined in the system.

Results and Discussion

Figures 3-5 show the results of the Matlab code for element lengths of 1, 0.5, and 0.25, respectively. The corresponding meshes for each element length are shown in the appendix. Each mesh was generated using the function *generateMesh* as shown in line 72 of the attached code.

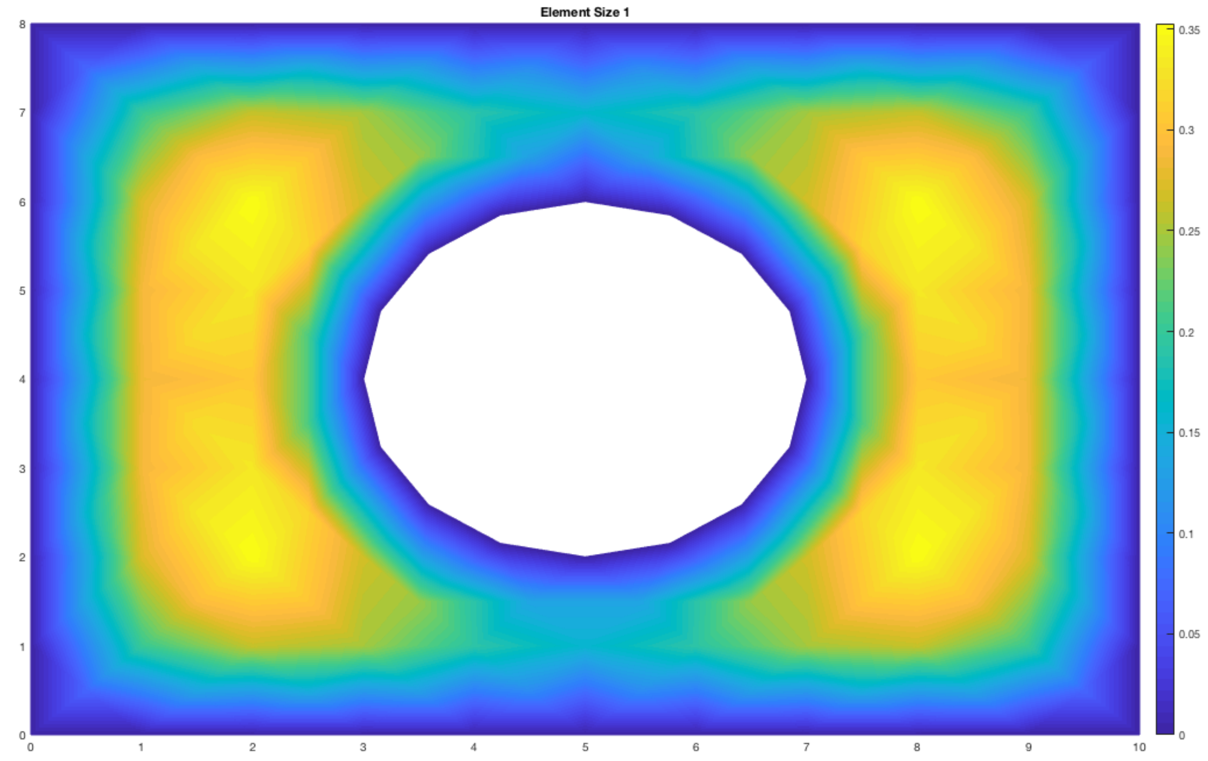
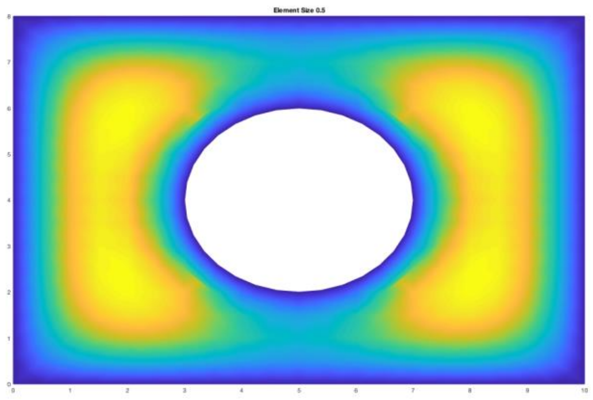


Figure 3: *Contour for Element Length 1*



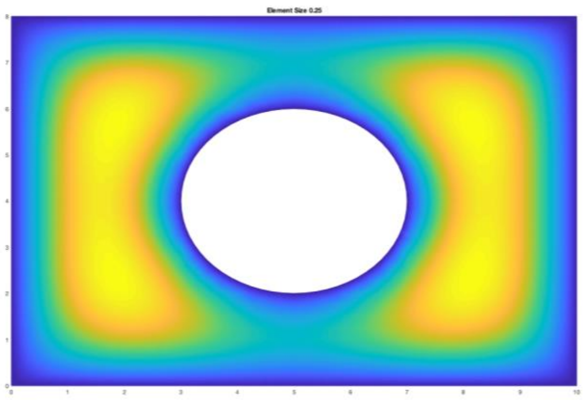


Figure 4: *Contour for Element Length 0.5*

Figure 5: *Contour for Element Length 0.25*

Figure 3 corresponds to the solution approximated using a step size of 1. The contours in figures 4 and 5 appear smoother since there are more approximated nodal values and the solution is interpolated over a smaller area.

A side view shown in figure 6 illustrates that the system exhibits the expected shape of laminar flow for a fluid.

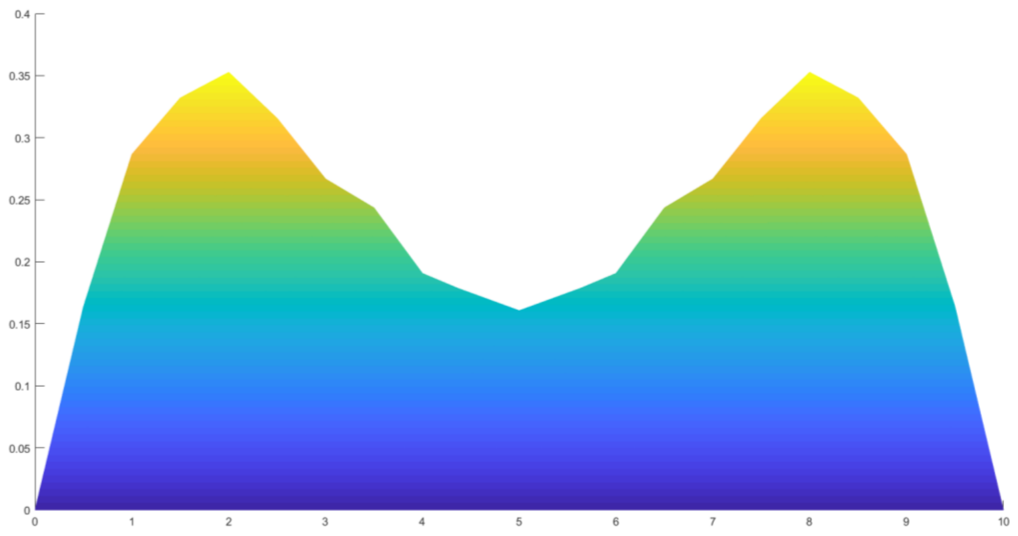


***Figure 6**: Side view of Element Length 1*

In order to prove the validity of the code, a simpler case was compared to a known exact solution. This is a case of fluid flow through 5X4 rectangle with the both the top and right sides having a boundary condition of u = 1 and the bottom and left side having a boundary condition of $u_y = 0$ and $u_x = 0$ respectively. The governing differential equation for this model is $\nabla^2 u - 1 = 0$ and both the analytical solution and the numerical approximation were evaluated for varying mesh sizes.

The analytical solution was computed as a 10-term series approximation of the equation shown below.

$$u = \sum_{n=1}^{10} \left[D_n \cos\frac{n\pi}{10} x \cosh\frac{n\pi}{10} y \right] + \frac{1}{2}(x^2 - 25) + 1 \tag{14}$$

$$\text{where} \qquad D_n = \frac{2}{n\pi \cosh\frac{2n\pi}{5}} \left[\frac{200}{(n\pi)^2} \sin\frac{n\pi}{2} \right]$$

Both the analytical solution and numerical approximation was calculated for 4 different mesh sizes of 1, 0.5, 0.25 and 0.1. The % error between the analytical solution and numerical approximation was calculated for every node and the average of the % error was calculated. Figure 7 shows the solutions for each mesh size.

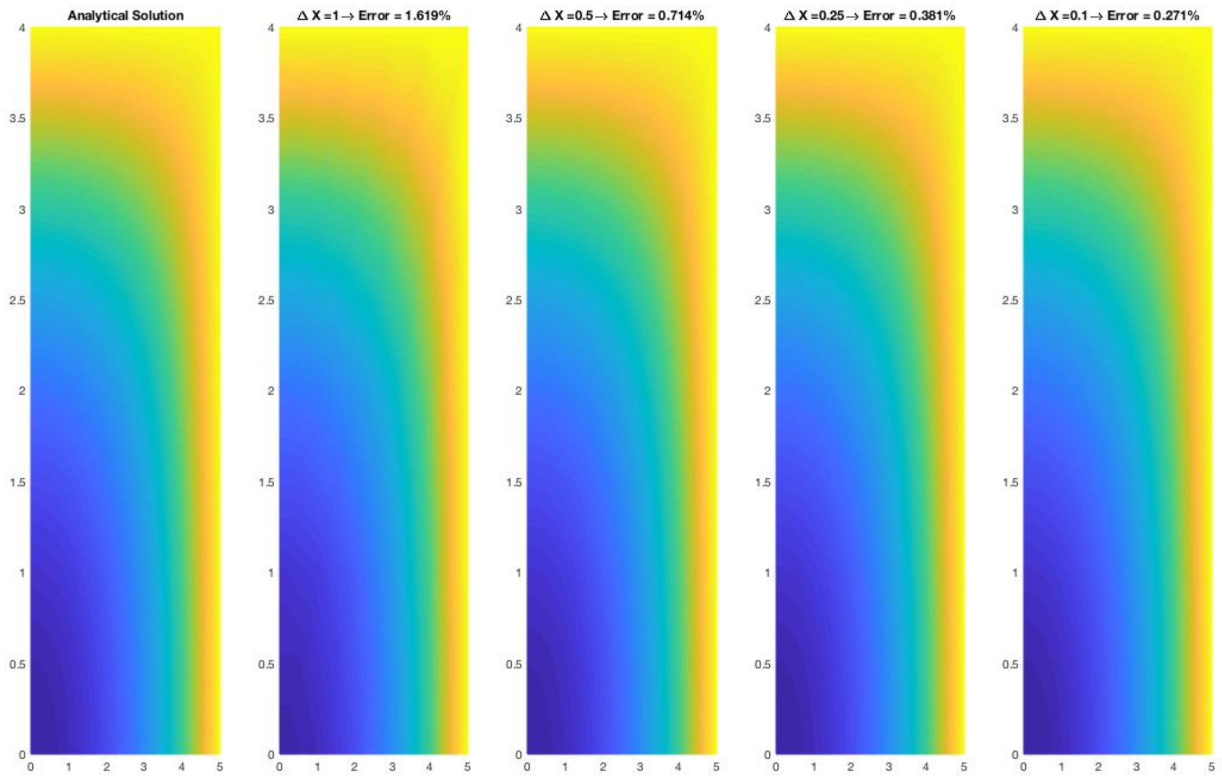


Figure 7*: Exact Solution and Approximations for a Simpler Case*

As seen in figure 7, going from an element length of 1 to 0.5 decreases the error by 55.8%. As the step size decreases, the percent reduction in error from one step size to the next decreases as well, showing convergence to the known exact solution.

Appendix

```
%% Setup Project
X = 10;  % X dimension of Rectangle
Y = 8;   % Y dimension of Rectangle
r = 2;   % Radius of Hole
L = 1; % Length of Elements

%% Mesh
out = generateMesh(X,Y,L,X/2,Y/2,r); % Meshing Function

XY = out.XY;               % Store XY Matrix from Function
CONN = out.CONN;           % Store CONN Matrix from Function
element = out.element;     % Store Element Data from Function
Hole = out.Hole;           % Store Hole Vertices created in Function

elementNumber = length(element);   % Number of Elements Modeled
nodeNumber = length(XY);           % Number of Nodes Modeled

AG = zeros(nodeNumber,nodeNumber); % Preallocate Stiffness Matrix
BG = zeros(nodeNumber,1);          % Preallocate Load Matrix

%% BC Data
[~,Hole] = ismember(Hole,XY,'rows'); % Find Nodes along Hole
LeftEdge = find(XY(:,1) == 0);       % Find Nodes along Left Edge
RightEdge = find(XY(:,1) == X);      % Find Nodes along Right Edge
TopEdge = find(XY(:,2) == Y);        % Find Nodes along Top Edge
BottomEdge = find(XY(:,2) == 0);     % Find Nodes along Bottom Edge
BCdata = [Hole; LeftEdge; RightEdge; TopEdge; BottomEdge];

%% Loop
for k  = 1:elementNumber       % Loop over all Elements
    currElement = element(k); % Load current Element
    [a,b,c,Ae] = simplexElements(currElement);  %Basis Fcn coeffs
    for i = 1:3
        for j = 1:3
           AL(i,j) = (b(i)*b(j) + c(i)*c(j));
        end
        BL(i) = 1/3*Ae;
    end
    node = element(k).CONN;    % Load Node Numbers for current Element
    for i = 1:3
        for j = 1:3
            AG(node(i),node(j)) = AG(node(i),node(j)) + AL(i,j);
        end
        BG(node(i)) = BG(node(i)) + BL(i);
    end
end

%% Boundary Update
for i = 1:numel(BCdata)
    row = BCdata(i);
    AG(row,:) = 0;
    AG(row,row) = 1;
    BG(row) = 0;
end

```

```
%% SOLVE
H = AG\BG;

%% Show Me
for i = 1:size(CONN,1)
    node1 = CONN(i,1);
    node2 = CONN(i,2);
    node3 = CONN(i,3);
    x = [XY(node1,1); XY(node2,1);XY(node3,1)];
    y = [XY(node1,2); XY(node2,2);XY(node3,2)];
    C = [H(node1);H(node2);H(node3)];
    patch(x,y,C,C,'EdgeColor','interp');
end
colorbar

%% Generate Mesh Function
function [out] = generateMesh(X,Y,L,xi,yi,r)
% Setup System
x = 0:X/L-1;
y = 0:Y/L-1;
% Removing elements inside the circle creates 8 sided polygon,
% with each side composed of r/L Element boundaries
n = 8*(r/L);

% Create HOLE
pgon = rotate(nsidedpoly(n,'center',[xi yi],'radius', r),(180/n),[xi
yi]);

% Create Larger Hole Mesh Refinement
pgon1 = nsidedpoly(n,'center',[xi yi],'radius', (r+L));

% Create Polyshapes
theta = 90;              % Rotate Element to get adjacent Element
centerx = [0 L 0.5*L];  %  x-coordinates of global Element 1
centery = [0 0 0.5*L];  %  y-coordinates of global Element 1

m = 1; n = 1;
for j = 1:length(y)
    posy = centery + L*y(j); % y-coordinates of current Element
    for i = 1:length(x)
        posx = centerx + L*x(i); % x-coordinates of current Element
        basePoly = polyshape(posx,posy);
        rotateVert = basePoly.Vertices(2,:);
        for k = 0:3
            currElement = rotate(basePoly,(k*theta),rotateVert);
            if overlaps(currElement,pgon1)
               chk = sum(isinterior(pgon1, currElement.Vertices));
               if chk == 3
                mesh(n).poly = currElement; n = n+1;
               elseif chk == 2
                mesh(n).poly = currElement; n = n+1;
               else
                element(m).poly = currElement; m = m+1;
               end
            else
```

```matlab
                    element(m).poly = currElement; m = m+1;
                end
            end
        end
end

% Mesh around Hole
reFine = union(subtract([mesh.poly],pgon));
TR = triangulation(reFine);
Tconn = TR.ConnectivityList;
TXY = round(TR.Points,3);

for i = 1:length(Tconn(:,1))
     for j = 1:3
         Nodes(j,:) = TXY(Tconn(i,j),:);
     end
   Telement(i).poly = polyshape(Nodes);
end

% Combine Elements
element= [element Telement];
XY = [0,0]; CONN = [];
for i = 1:length(element)
    currVerts = round(element(i).poly.Vertices,3);
    idx = ismember(currVerts,XY,'rows');
    XY = [XY; currVerts(idx == 0,:)];
    [~,Locb] = ismember(currVerts,XY,'rows');
    CONN = [CONN; Locb'];
    element(i).XY = currVerts;     % Store  XY  data in each ELEMENT
    element(i).CONN = Locb';       % Store CONN data in each ELEMENT
end

% Gather Data
out.element = element;
out.XY = XY;
out.CONN = CONN;
out.Hole = round(pgon.Vertices,3);
end

%% Simplex Elements Function
function [a, b, c, Ae] = simplexElements(element)
Ae = area(element.poly);
X = element.XY(:,1);
Y = element.XY(:,2);

a(1) = 1/2/Ae*(X(2)*Y(3)-X(3)*Y(2));
a(2) = 1/2/Ae*(X(3)*Y(1)-X(1)*Y(3));
a(3) = 1/2/Ae*(X(1)*Y(2)-X(2)*Y(1));

b(1) = -1/2/Ae*(Y(3)-Y(2));
b(2) = -1/2/Ae*(Y(1)-Y(3));
b(3) = -1/2/Ae*(Y(2)-Y(1));

c(1) = 1/2/Ae*(X(3)-X(2));
c(2) = 1/2/Ae*(X(1)-X(3));
```

```
c(3) = 1/2/Ae*(X(2)-X(1));
end

%% Plot Mesh with Nodal/Elemental Numbering
function plotMesh(XY,CONN)
figure; hold on
for i = 1:size(CONN(:,1))
    Node1 = XY(CONN(i,1),:);
    Node2 = XY(CONN(i,2),:);
    Node3 = XY(CONN(i,3),:);
    plot([Node1(1),Node2(1)],[Node1(2),Node2(2)])
    plot([Node2(1),Node3(1)],[Node2(2),Node3(2)])
    plot([Node3(1),Node1(1)],[Node3(2),Node1(2)])
    node = [Node1;Node2;Node3];
    x = node(:,1);
    y = node(:,2);
    poly = polyshape(x,y);
    [x,y] = centroid(poly);
    text(x,y,int2str(i),'fontsize',10,'color','b',...
        'HorizontalAlignment','center') ;
end
for i = 1:size(XY(:,1))
    if ismember(i,CONN)
    plot(XY(i,1),XY(i,2),'ow','MarkerSize',15,'MarkerFaceColor','w')
    text(XY(i,1),XY(i,2),int2str(i),'fontsize',10,'color','r',...
        'HorizontalAlignment','center');
    end
end
end
```

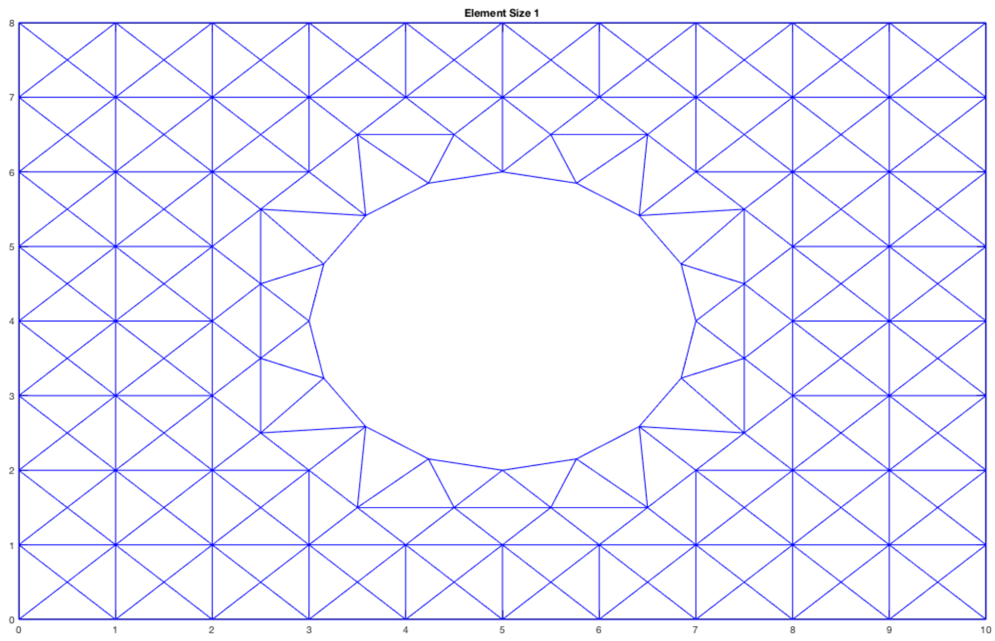


Figure 8: *Mesh Element Length 1*

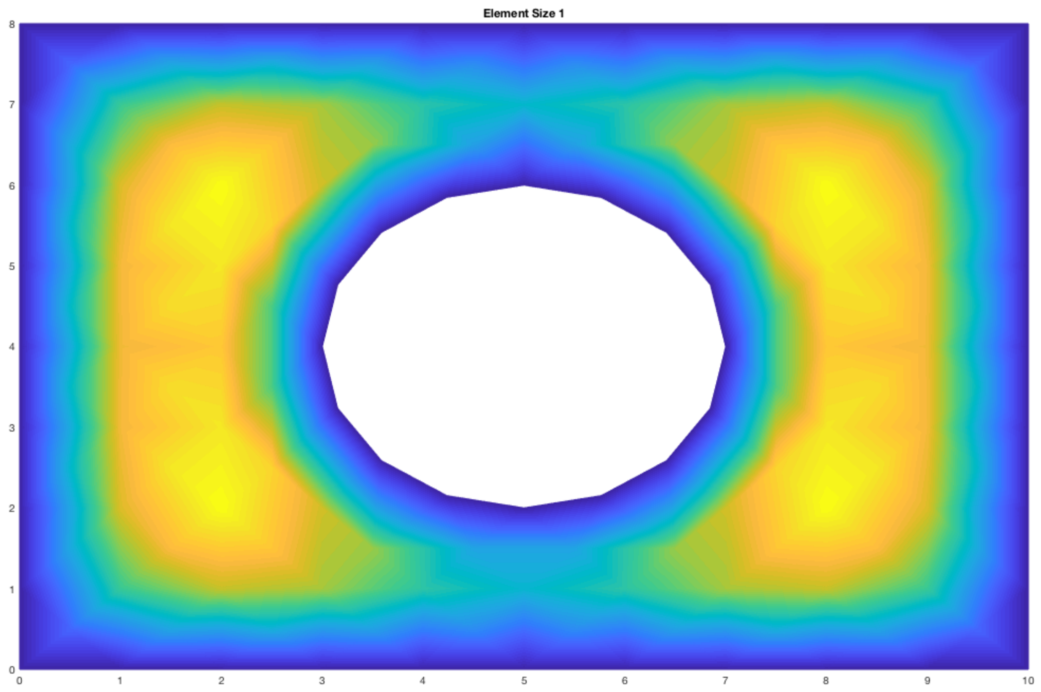


Figure 9: *Contour Element Length 1*

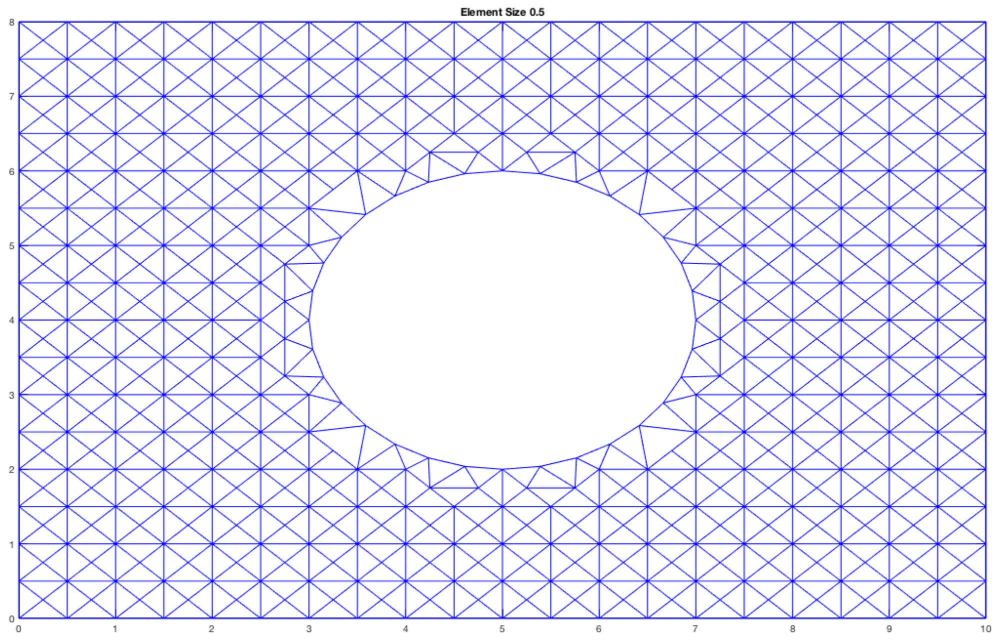


Figure 10: *Mesh Element Length 0.5*

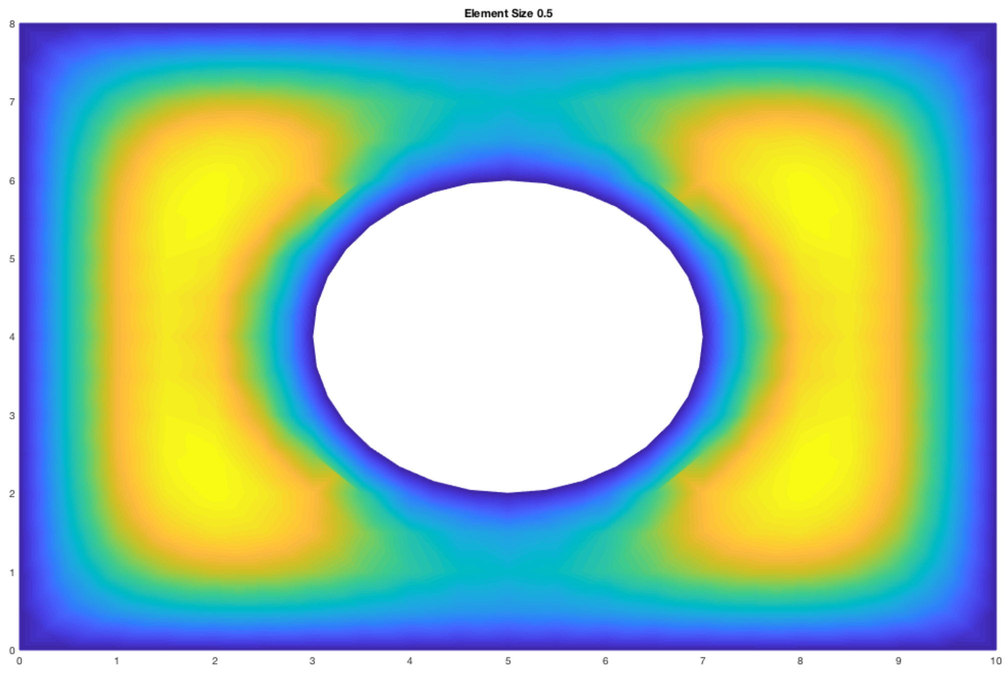


Figure 11: *Mesh Element Length 0.5*

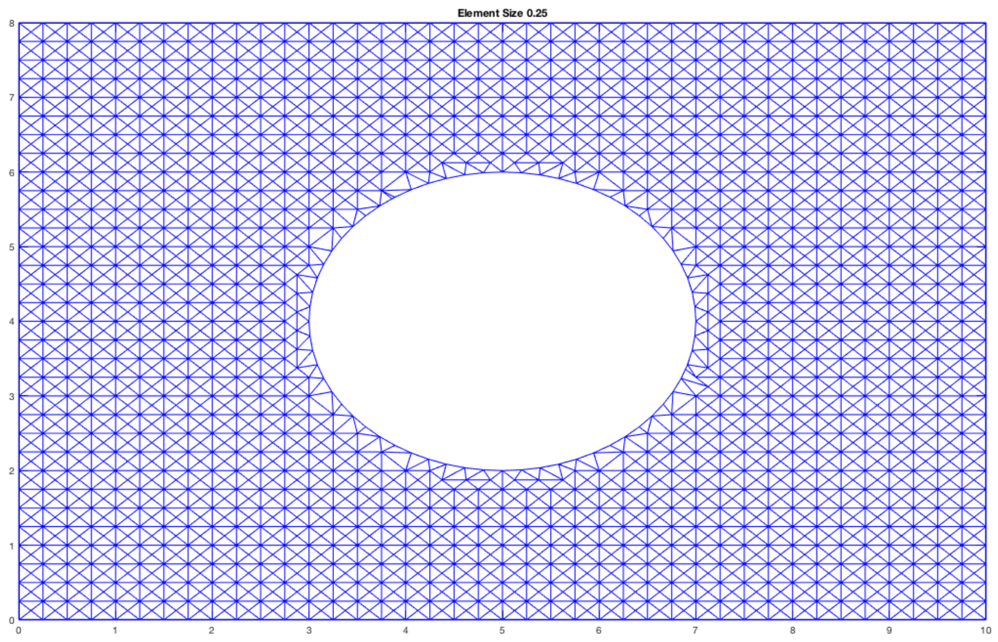


Figure 12: *Mesh Element Length 0.25*

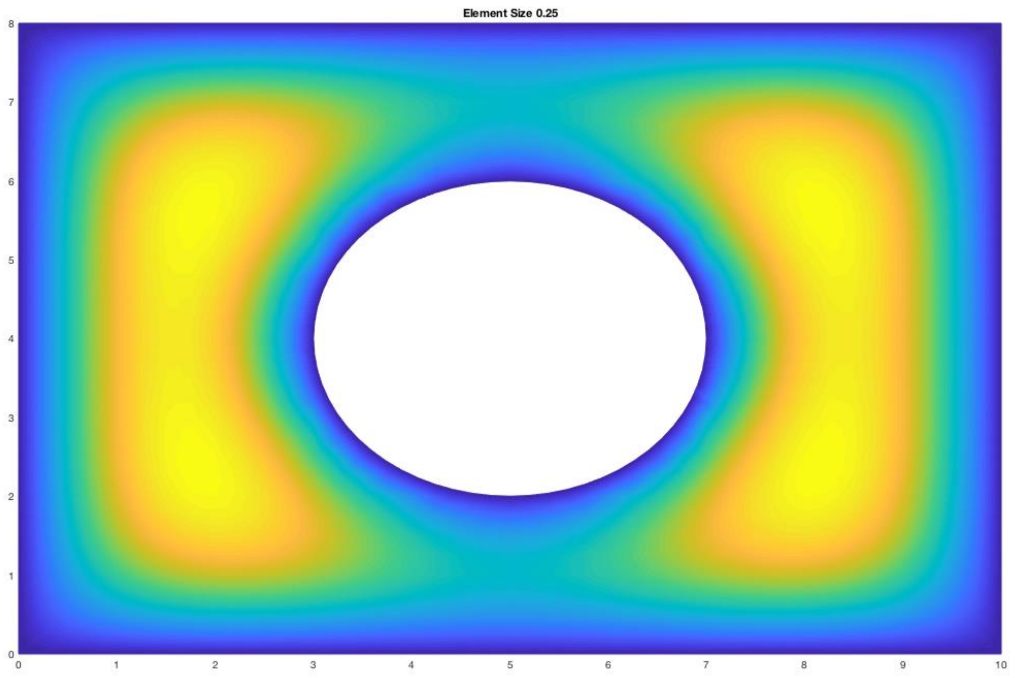


Figure 13: *Mesh Element Length 0.25*