```
% Final Script
% Because of Symmetric arrangement, I am only considering top-right part
clear all; clc

%% Basic Inputs

% a & b Sizes
tot_stress = [];
% for b = linspace(0.01,0.2,10)
a = 0.1;t = 1;b=0.01;sigma = 200e6;
iter = 15;
%iter = 10;
tot_disp = [];
%Mesh Density Gradian
gradient = 1.15;
%for iter = 2:max_iter
    % Mesh Sizings
    nx = iter;
    ny = iter;

%% Setting up the Geometry

ellipse_points = [a*cos(linspace(0,pi/2,ny+nx-1));b*sin(linspace(0,pi/2,ny+nx-1))];
top_points = [tan(linspace(pi/4,0,nx));zeros(1,nx)+1];
right_points = [zeros(1,ny)+1; tan(linspace(0,pi/4,ny))];

% Mesh Generation
    X=zeros(nx,ny);
    Y=zeros(nx,ny);

% Generate Node Coordiantes
    R = nx+ny;
    for i = 1:ny
        X(i,1) = ellipse_points(1,i);
        X(i,2) = X(i,1)+(right_points(1,i)-ellipse_points(1,i)) *
((1-gradient)/(1-gradient^(R-1)));
        for j = 3:R
            X(i,j) = X(i,j-1) + (X(i,j-1)-X(i,j-2))*gradient;
        end
        X(i,R) = right_points(1,i);
    end
    for i = 1:nx
        X(i+ny-1,1) = ellipse_points(1,i+ny-1);
        X(i+ny-1,2) = X(i+ny-1,1)+(top_points(1,i)-ellipse_points(1,i+ny-1)) *
((1-gradient)/(1-gradient^(R-1)));
        for j = 3:R
            X(i+ny-1,j) = X(i+ny-1,j-1) + (X(i+ny-1,j-1)-X(i+ny-1,j-2))*gradient;
        end
        X(i+ny-1,R) = top_points(1,i);
    end
    for i = 1:nx
        Y(i,1) = ellipse_points(2,i);
        Y(i,2) = Y(i,1)+(right_points(2,i)-ellipse_points(2,i)) *
((1-gradient)/(1-gradient^(R-1)));
        for j = 3:R
            Y(i,j) = Y(i,j-1) + (Y(i,j-1)-Y(i,j-2))*gradient;
        end
        Y(i,R) = right_points(2,i);
    end
    for i = 1:ny
        Y(i+nx-1,1) = ellipse_points(2,i+nx-1);
        Y(i+nx-1,2) = Y(i+nx-1,1)+(top_points(2,i)-ellipse_points(2,i+nx-1)) *
((1-gradient)/(1-gradient^(R-1)));
        for j = 3:R
            Y(i+nx-1,j) = Y(i+nx-1,j-1) + (Y(i+nx-1,j-1)-Y(i+nx-1,j-2))*gradient;
        end
        Y(i+nx-1,R) = top_points(2,i);
    end

% Calculating Mesh Details
[p,q] = size(X);
total_elements = (p-1)*(q-1);
total_nodes = p*q;
bottom_nodes = 1:q;
top_nodes = linspace(ceil(p/2)*q,p*q,floor(q/2));
side_nodes = bottom_nodes + (p-1)*q;

% Calculating the Coordinates for all nodes
all_nodes = zeros(total_elements,4);
individual_nodes = [1,2,2+q,1+q];
n = 1;
for i = 1:p-1
    for j = 1:q-1
        all_nodes(n,:) = individual_nodes;
        element_X_coord(n,:) = [X(i,j),X(i,j+1),X(i+1,j+1),X(i+1,j)];
        element_Y_coord(n,:) = [Y(i,j),Y(i,j+1),Y(i+1,j+1),Y(i+1,j)];
        individual_nodes = individual_nodes+1;
        n = n+1;
    end
    individual_nodes = individual_nodes+1;
end

%% FEA Part

% Global Stiffness Matrix
Kg = zeros(2*total_nodes);
for i = 1:total_elements
    L = gather(total_nodes,all_nodes,i);
    Kg = Kg + L'*stiffness(element_X_coord(i,:),element_Y_coord(i,:))*L*t;
end
```

```
% Formulaitng the Force Vecor
F = zeros(2*total_nodes,1);

for i = 1:length(top_nodes)-1
    ele_length = X(floor(top_nodes(i)/q), q) - X(floor(top_nodes(i+1)/q), q);
    F_curr = (1/2)*t*abs(ele_length)*sigma;
    F(2*top_nodes(i),:) = F(2*top_nodes(i),:) + F_curr;
    F(2*top_nodes(i+1),:) = F(2*top_nodes(i+1),:) + F_curr;
end

% Applying Boundary Conditions
bound_dof = [];
for i = 1:length(bottom_nodes)
    bound_dof = [bound_dof 2*bottom_nodes(i) 2*side_nodes(i)-1];
end

F_new = F;    Kg_new = Kg;   l = [];
bound_dof_new = sort(bound_dof);
for i = 1:length(bound_dof)
        F_new(bound_dof_new(i)) = [];
        Kg_new(bound_dof_new(i),:) = []; Kg_new(:,bound_dof_new(i)) = [];
        bound_dof_new = bound_dof_new-1;
end

disp_new = linsolve(Kg_new,F_new);
n = 0;
disp = zeros(2*total_nodes,1);

counter = 1;
for i =  1:length(disp)
    if sum(i == bound_dof) == 1
        disp(i) = 0;
    else
        disp(i) = disp_new(counter);
        counter = counter + 1;
    end
end

R = Kg*disp;
dx = disp(1:2:end);    dy = disp(2:2:end);
tot_def = sqrt(dx.^2 + dy.^2);

loc=1;
for i = 1:p
    for j =1:q
        X_mod(i,j) = X(i,j)+ 100*dx(loc); % Applied a Scaling Factor of 100
        Y_mod(i,j) = Y(i,j)+ 100*dy(loc); % Applied a Scaling Factor of 100
        loc = loc+1;
    end
```

```
end


Sg_elements = zeros(total_elements,2);
Sg_nodes = zeros(total_nodes,2);
node_counter = zeros(total_nodes,1);
for i = 1:total_elements
    element_nodes = all_nodes(i,:);
    Q = zeros(8,1);
    for nodes =1:4
        Q(2*nodes-1) = dx(element_nodes(nodes));
        Q(2*nodes) = dy(element_nodes(nodes));
    end
    Sg_elements(i,:) = stress(element_X_coord(i,:),element_Y_coord(i,:),Q);
    for j = 1:4
        curr_node = all_nodes(i,j);
        Sg_nodes(curr_node,1) = Sg_nodes(curr_node,1) + Sg_elements(i,1);
        Sg_nodes(curr_node,2) = Sg_nodes(curr_node,2) + Sg_elements(i,2);
        node_counter(curr_node) = node_counter(curr_node) + 1;
    end
end

Sg_nodes = Sg_nodes./node_counter;

max(Sg_nodes(:,2))
%% Contours and Plots

%if iter == 20
    figure(1)
    hold on
    for i = 1:ny+nx-1
        plot(X(i,:),Y(i,:),'k')
        plot(X_mod(i,:),Y_mod(i,:),'b')
    end
    plot(X,Y,'k')
    plot(X_mod,Y_mod,'b')
    xlim([-0.2 1.2])
    ylim([-0.2 1.2])
    title('Mesh with & w/o Deflection(100x)')
    hold off

    von_misses = transpose(reshape(Sg_nodes(:,1),q,p));
    figure(4)
    contourf(X,Y,von_misses,20)
    xlim([-0.2 1.2])
    ylim([-0.2 1.2])
    title('Von-Misses Stress')
    colorbar()

    principal = transpose(reshape(Sg_nodes(:,2),q,p));
    figure(5)
    contourf(X,Y,principal,50)
    xlim([0 0.25])
    ylim([0 0.25])
    title('Principle Y Stress')
    colorbar()

% tot_disp = [tot_disp sum(abs(disp))];
 tot_stress = [tot_stress (Sg_nodes(1,2))];
%  end


%
% Plot for Testing Convergence
% figure(2)
%
plot((tot_disp(2:length(tot_disp))-tot_disp(1:length(tot_disp)-1))./tot_disp(1:lengt
h(tot_disp)-1))
% xlim([0 max_iter+2])
% ylim([0 2])
% xlabel('Nx & Ny')
% ylabel('% change')
% title('Convergence for sum of absolute of all node Displacements')

% Plot for Different Values of b
% figure(6)
% plot(linspace(0.01,0.2,10),tot_stress)
% title('Varying values of b')
% xlabel('value of b in [m]')
% ylabel('stress in [Pa]')

%% Functions for Stiffness & Gather Matrix Calculations

    function K = stiffness(X,Y)
    E = 210e9; % [Pa]
    nu = 0.3;
    D = (E/(1-nu^2))*[1 nu 0; nu 1 0; 0 0 (1-nu)/2];
    coord = [X',Y'];
    K = zeros(8,8);
    for i = 1:2
        for j = 1:2
            eta = (2*i-3)/sqrt(3);
            zeta = (2*j-3)/sqrt(3);
            J = (1/4)*[eta-1 1-eta 1+eta -eta-1; zeta-1 -zeta-1 1+zeta
1-zeta]*coord;
            H = (1/4)*[eta-1 1-eta 1+eta -eta-1; zeta-1 -zeta-1 1+zeta 1-zeta];
            H = J\H;
            H = [H(1,1) 0 H(1,2) 0 H(1,3) 0 H(1,4) 0; 0 H(2,1) 0 H(2,2) 0 H(2,3) 0
H(2,4); H(2,1) H(1,1) H(2,2) H(1,2) H(2,3) H(1,3) H(2,4) H(1,4)];
            K = K + det(J)*H'*D*H;
        end
    end
    end

    function S = stress(X,Y,Q)
    E = 210e9; % [Pa]
    nu = 0.3;
    D = (E/(1-nu^2))*[1 nu 0; nu 1 0; 0 0 (1-nu)/2];
    coord = [X',Y'];
    stress_temp = zeros(3,1);
    for i = 1:2
        for j = 1:2
            eta = (2*i-3)/sqrt(3);
            zeta = (2*j-3)/sqrt(3);
            J = (1/4)*[eta-1 1-eta 1+eta -eta-1; zeta-1 -zeta-1 1+zeta
1-zeta]*coord;
            H = (1/4)*[eta-1 1-eta 1+eta -eta-1; zeta-1 -zeta-1 1+zeta 1-zeta];
            H = J\H;
            H = [H(1,1) 0 H(1,2) 0 H(1,3) 0 H(1,4) 0; 0 H(2,1) 0 H(2,2) 0 H(2,3) 0
H(2,4); H(2,1) H(1,1) H(2,2) H(1,2) H(2,3) H(1,3) H(2,4) H(1,4)];
            stress_temp = stress_temp + 0.25*D*H*Q; % Sigma XX, Sigma YY, Sigma XY
        end
    end
    stress_xx = stress_temp(1);
    stress_yy = stress_temp(2);
    stress_xy = stress_temp(3);
    stress_vm = sqrt(stress_xx^2 + stress_yy^2 + 3*stress_xy^2 -
stress_xx*stress_yy);
    S = [stress_vm stress_yy];
end


    function L = gather(total_nodes,elements,n)
    L = zeros(8,2*total_nodes);
    L(1:2,2*elements(n,1)-1:2*elements(n,1)) = eye(2);
    L(3:4,2*elements(n,2)-1:2*elements(n,2)) = eye(2);
    L(5:6,2*elements(n,3)-1:2*elements(n,3)) = eye(2);
    L(7:8,2*elements(n,4)-1:2*elements(n,4)) = eye(2);
    end
```