Table of Contents

Part 1 1
Part 2 10
Part 3 17

Part 1

```
clc
clear
close all
%Setting Constants
mElec = 9.11e-31; % Electron rest mass (kg)
mEff = 0.26*mElec; % Effective mass (kg)
kb = 1.381e-23; % Boltzmann's Constant (J/K)
T = 300; % Room Temperature (K)
L = 200e-9; % Length (m)
W = 100e-9; % Width (m)
q = 1.60218e-19; % Elemntary Charge (C)

% a) Electric Field Quiver
Vvec = linspace(0.1, 0, 200);
Vmat = zeros(100, 200);
for k = 1:100
   Vmat(k,:) = Vvec;
end

figure(1)
surf(Vmat)
xlabel('Length')
ylabel('Width')
zlabel('Voltage')
title('Voltage Over The Semiconductor Area')
grid on

[Ex,Ey] = gradient(-Vmat, 1e-9);
figure(2)
quiver(Ex, Ey)
xlabel('Length')
ylabel('Width')
title('Electric Field Quiver Across The Semiconductor Area')
xlim([0,200])
ylim([0,100])

% Looking at the quiver, it is obvious that the electric field across
 the
% region is constant. del(V) = -Ed, since d and del(V) are constant
 (200nm and 0.1 V respectively), the E
% field stays the same
Ehor = Ex(1, 1); % Horizontal E Field
```

```
Ever = Ey(1, 1); % Vertical E Field (Should be zero for the 0.1V on
 the x axis case)
fprintf('The E field is constant across the semiconductor and is equal
 to %f V/m in the  x direction (no y component)\n\n', Ehor)

% The results for this section are consistent with the theoretical
% expectations, Since the voltage of across the semi conductor changes
% linearly from 0.1 and only in the X direction, it is expected that
 the
% voltage matrix looks like a ramp. The gradient of the voltage matrix
 is
% constant which results in a constant E Field in the direction of the
% change in V (x direction)

% b) Calculated Force
Fx = q*Ehor; % Force on the electrons (in Newtons)
Fy = q*Ever; % Support for forces generated by E fields with vertical
 components
fprintf('Since E field is only in the x direction, the force on each
 electron is %f e-15 N in the x direction (no y component)\n\n',
 Fx*1e15)
% The force was derived from the electric field using the equation F =
 Eq
% where q is the charge of an electron.

% c) 2-D Particle trajectories and the calculated Acceleration
% To solve for the accelerations, newton's law F = ma can be
 rearranged to
% a = F/m and since we already derived F from part b, the acceleration
 can
% be easily calculated

ax = Fx/mEff; % Horizontal Acceleration
ay = Fy/mEff; % Vertical Acceleration

fprintf('There is no vertical component to the acceleration as the
 force is horizontal. The acceleration is %f E15 m/s^2 x direction (no
 y component)\n\n', ax*1e-15)

% Using the modified code from assignment 1 to generate the 2-D plot
 of
% electron trajectories

% Finding the thermal velocity
Vth = sqrt((2*kb*T)/mEff); %Thermal Velocity in m/s
mt = 0.2e-12; % mean time (s)

% Modeling the electron paths

numPar = 10; % Number of particles

% Assigning particle positions
posX = L.*rand(numPar,2);
posY = W.*rand(numPar,2);
```

```
posX(:,1) = posX(:,2);
posY(:,1) = posY(:,2);

spacialStep = sqrt(L^2+W^2)/1000;
stepTime = spacialStep/Vth;

% Assigning each particle a random velocity (acceleration will be
 taken into account during the iteration)
Vx = randn(numPar,2)*sqrt((kb*T)/mEff);
Vy = randn(numPar,2)*sqrt((kb*T)/mEff);

% Displacement per step of each electron
dispX = stepTime*Vx(:,1);
dispY = stepTime*Vy(:,1);

colors = rand(numPar,3);

pScatter = 1 - exp(-stepTime/mt); % Probability of an electron
 scattering

% Positions before collision
collX = posX(:,1);
collY = posY(:,1);

ic = 0; % iteration counter for animated plot

for i = 1:1000
    % Adding the extra velocity from the acceleration (x only)
    Vx = Vx + ax*stepTime;
    Vy = Vy + ay*stepTime;
    for j = 1:numPar
        if(rand < pScatter)
            % Incrementing the collision counter and generating new
            % velocity/displacment values
            angle = 2*pi.*rand(1);
            Vx(j,:) = randn(1)*sqrt((kb*T)/mEff) + ax*stepTime;
            Vy(j,:) = randn(1)*sqrt((kb*T)/mEff) + ay*stepTime;
            dispX(j,:) = stepTime*Vx(j,1);
            dispY(j,:) = stepTime*Vy(j,1);
            collX = posX(:,1);
            collY = posY(:,1);
        end

        if (posX(j,1)+dispX(j) > L)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)-L;
        elseif (posX(j,1)+dispX(j) < 0)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)+L;
        else
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j);
        end
```

```
            if ((posY(j,1)+dispY(j) > W)|| (posY(j,1)+dispY(j) < 0))
                dispY(j) = -dispY(j);
                posY(j,2) = posY(j,1)+dispY(j);
            else
                posY(j,2) = posY(j,1)+dispY(j);
            end
        end
        if (ic == 0)
           figure(3);
           scatter(posX(:, 2),posY(:,2),1,colors(:,1));
           hold on;
           title('2-D Particle pathing');
           xlabel('Length');
           ylabel('Width');
           xlim([0 L]);
           ylim([0 W]);
        elseif (ic < 1000)
           title('2-D Particle pathing');
           scatter(posX(:, 2),posY(:,2),1,colors(:,1));
           hold on;
        else
           title('2-D Particle pathing');
           scatter(posX(:, 2),posY(:,2),1,colors(:,1));
           hold off;
        end
        pause(0.001);
        posX(:,1) = posX(:,2);
        posY(:,1) = posY(:,2);
        ic = ic + 1;
    end

    % d) Cuurrent density vs Time
    % J = n*(q/density)*mean(Vn), We can get the drift current density
     from
    % this equation using the average velocity

    conc = 10^19; % concentration of electrons per m^2
    numPar2 = 10000;
    Vx = randn(numPar2,2)*sqrt((kb*T)/mEff);
    Vy = randn(numPar2,2)*sqrt((kb*T)/mEff);
    posX = L.*rand(numPar2,2);
    posY = W.*rand(numPar2,2);
    posX(:,1) = posX(:,2);
    posY(:,1) = posY(:,2);
    dispX = stepTime*Vx(:,1);
    dispY = stepTime*Vy(:,1);
    ic = 0;
    avgVx = 0;
    avgVy = 0;

    for i = 1:1000
        Vx = Vx + ax*stepTime;
        Vy = Vy + ay*stepTime;
```

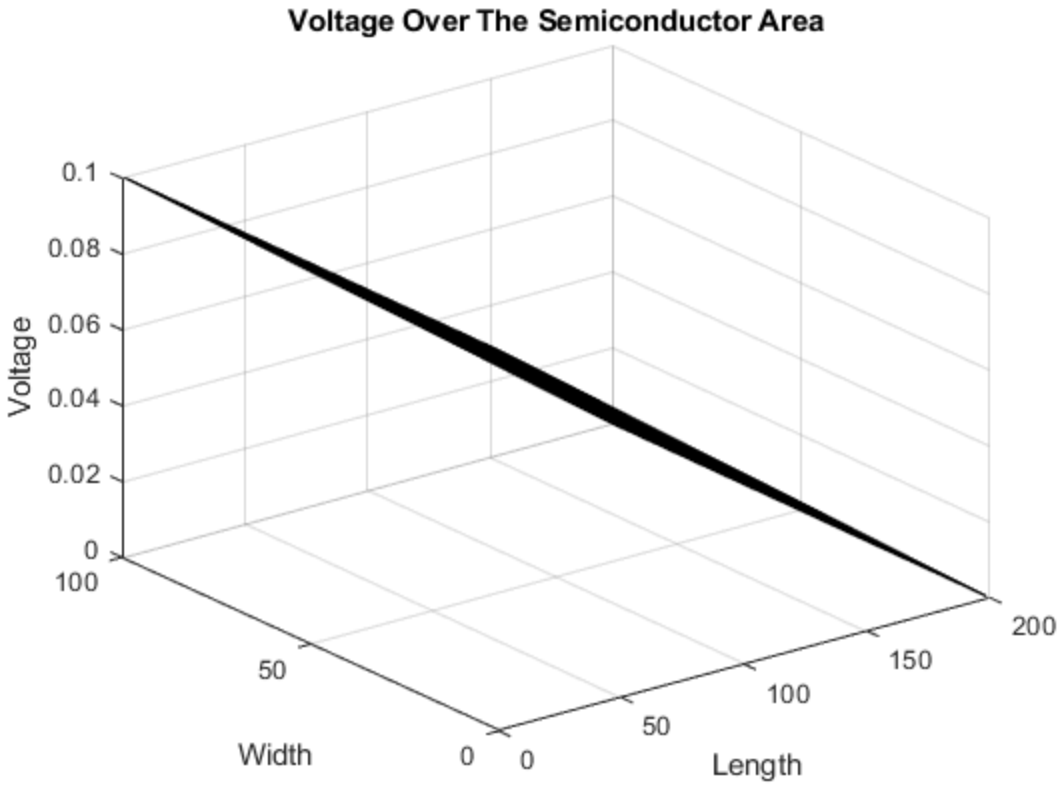
```
    avgVx(i) = mean(Vx(:,2));
    avgVy(i) = mean(Vy(:,2));

    for j = 1:numPar2
        if(rand < pScatter)
            Vx(j,:) = randn(1)*sqrt((kb*T)/mEff) + ax*stepTime;
            Vy(j,:) = randn(1)*sqrt((kb*T)/mEff) + ay*stepTime;
            dispX(j,:) = stepTime*Vx(j,1);
            dispY(j,:) = stepTime*Vy(j,1);
            collX = posX(:,1);
            collY = posY(:,1);
        end
        if (posX(j,1)+dispX(j) > L)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)-L;
        elseif (posX(j,1)+dispX(j) < 0)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)+L;
        else
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j);
        end

        if ((posY(j,1)+dispY(j) > W)|| (posY(j,1)+dispY(j) < 0))
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        else
            posY(j,2) = posY(j,1)+dispY(j);
        end
    end
    timeVec(i) = stepTime*i; % time vector for plot
end
J = W*q*conc.*avgVx;
figure(4)
plot(timeVec, J)
title('Current Density vs Time')
xlabel('Time')
ylabel('Current Density')

% The current density starts at 0 at t = 0 and increases rapidly
 before
% converging at a specific value (around 0.01 A/m for this case)

% e) Electron Dennsity map and Temperature plot

% Electron Density Map
figure(5)
hist3([posX(:,1) posY(:,1)],[20 10])
xlabel('x-axis')
ylabel('y-axis')
zlabel('Number of electrons')
title('Electron Density Map')

% Getting the temperature and kinetic energies
```

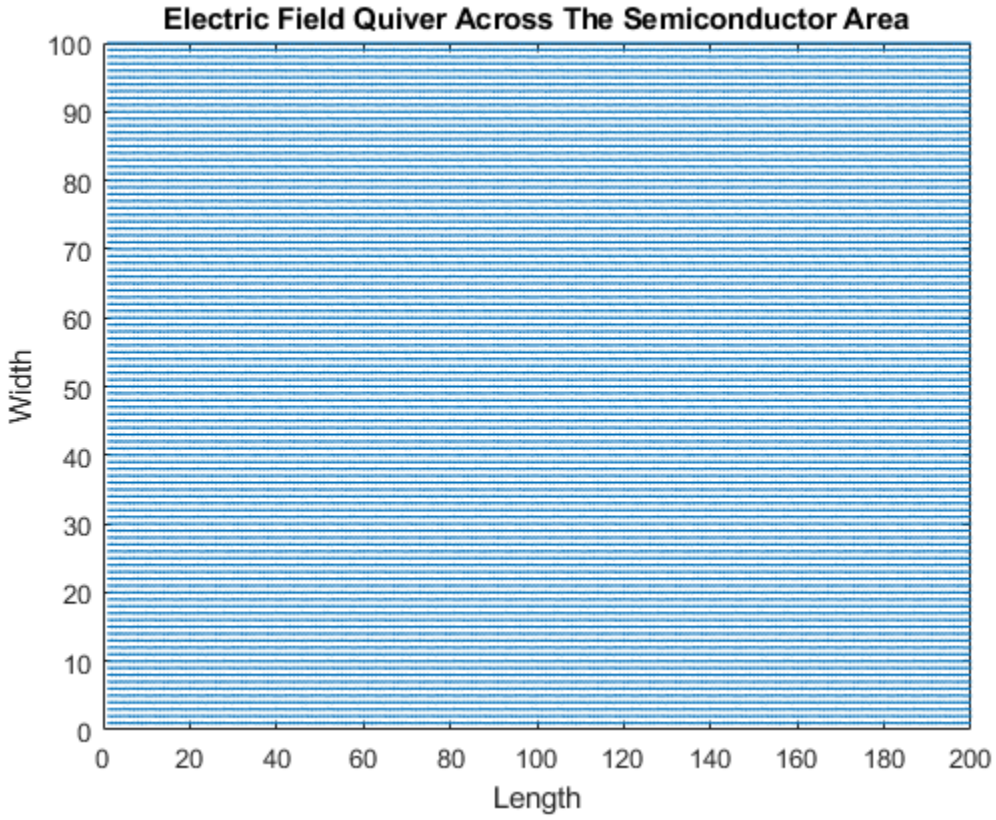
```
Vmag = sqrt(Vx(:,1).^2 + Vy(:,1).^2); % Magnitude of particle
 velocities
KE = 0.5*mEff.*Vmag.^2; % Kinetic Energy
T_each = KE./kb; %Temperature of each electron

% Rounding and scaling particle positions to use them as coordinates
 for
% the temperature map
posX_nm = round(posX(:,1).*1e9,-1)./10;
posY_nm = round(posY(:,1).*1e9,-1)./10;

temps = zeros(round(W/1e-9)/10,round(L/1e-9)/10);

for i = 1:numPar2
    posX_nm = round(posY(i,1).*1e9,-1)/10;
    posY_nm = round(posX(i,1).*1e9,-1)/10;
    if(posX_nm <= 0 || posY_nm <= 0)
        posX_nm = 1;
        posY_nm = 1;
    end
    temps(posX_nm, posY_nm) = temps(posX_nm,posY_nm) + T_each(i); %
 Temperature matrix
end

numPerBar = hist3([posX(:,1) posY(:,1)],[10 20]); % Number of
 electrons per bar in the temperature map

for i = 1:10
    for j = 1:20
        if(numPerBar(i,j) == 0)
            temps(i,j) = 0;
        else
            temps(i,j) = temps(i,j)/numPerBar(i,j);
        end
    end
end
temps(1,1) = 0;
figure(6)
bar3(temps)
xlabel('x-axis')
ylabel('y-axis')
zlabel('Temperature (K)')
title('Temperature Map')
```

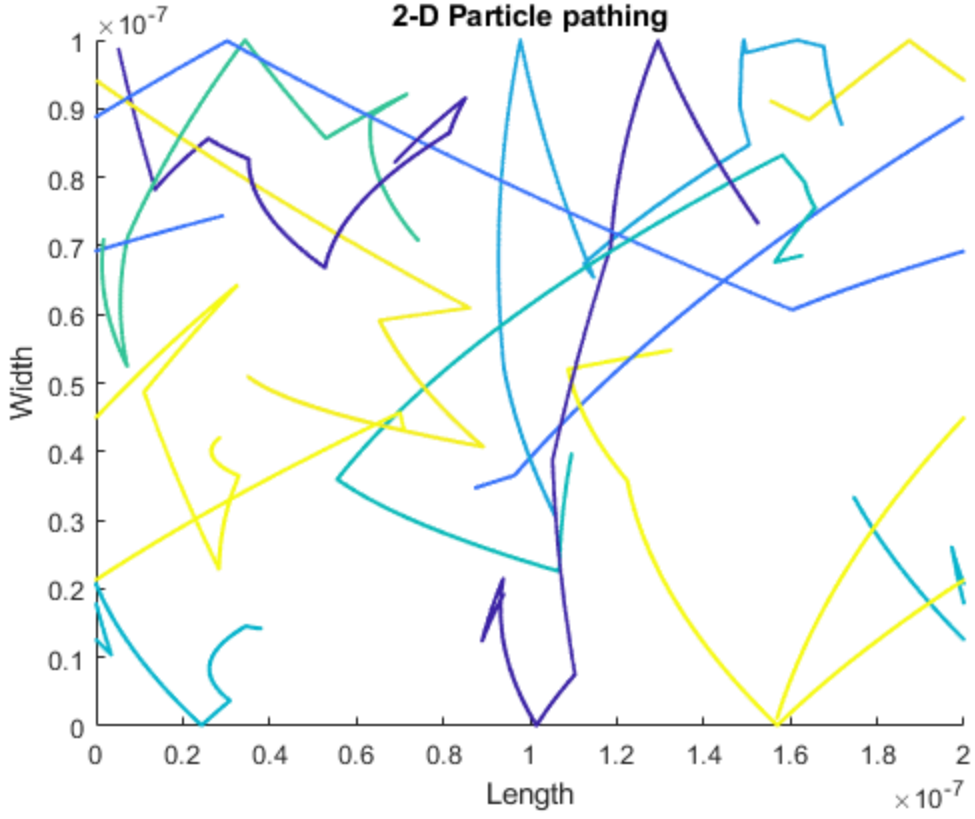
The E field is constant across the semiconductor and is equal to 502512.562814 V/m in the x direction (no y component)

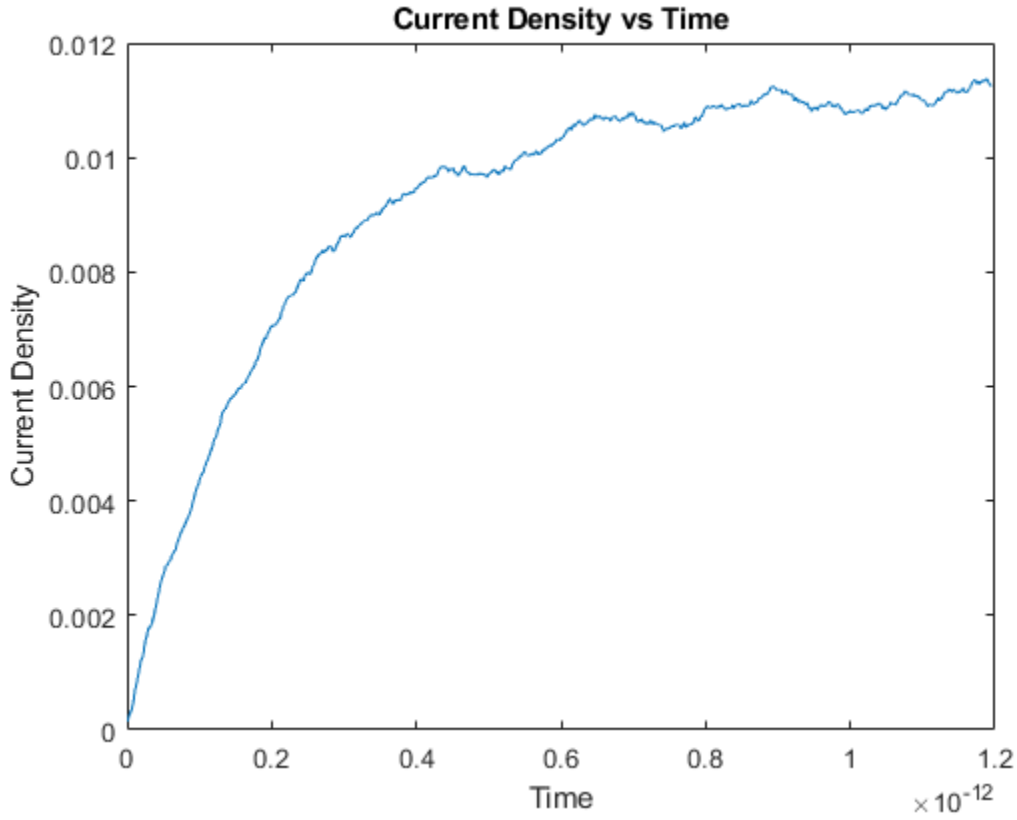
Since E field is only in the x direction, the force on each electron is 80.511558 e-15 N in the x direction (no y component)

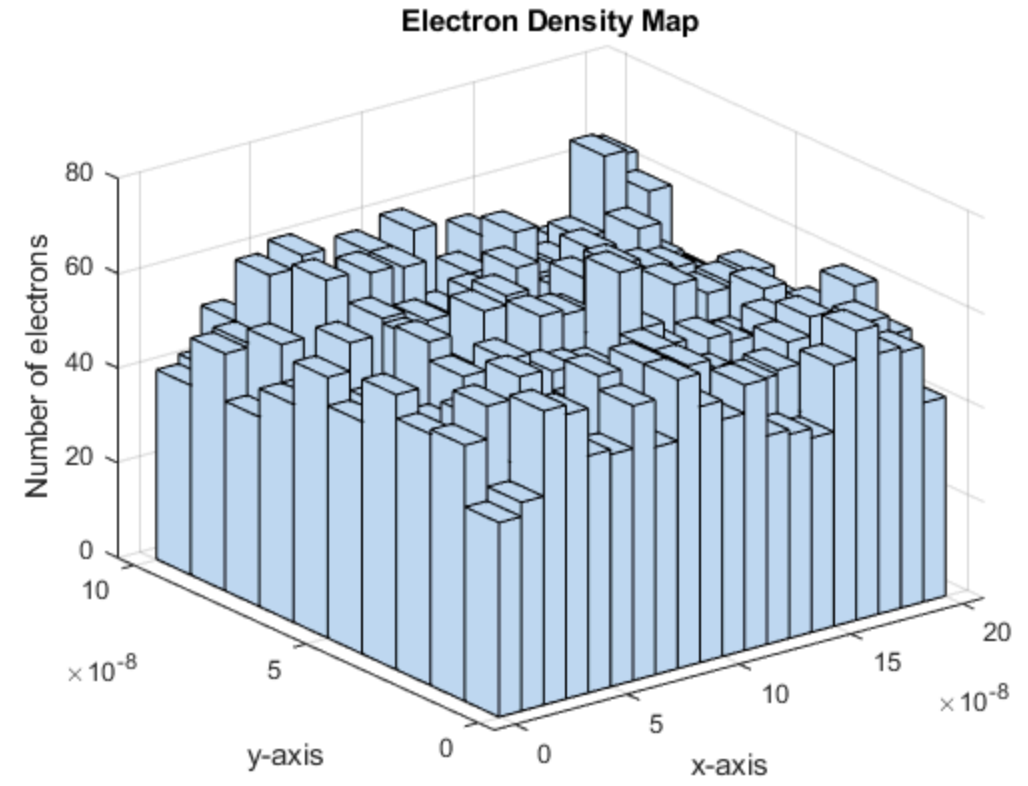
There is no vertical component to the acceleration as the force is horizontal. The acceleration is 339.912006 E15 m/s^2 x direction (no y component)

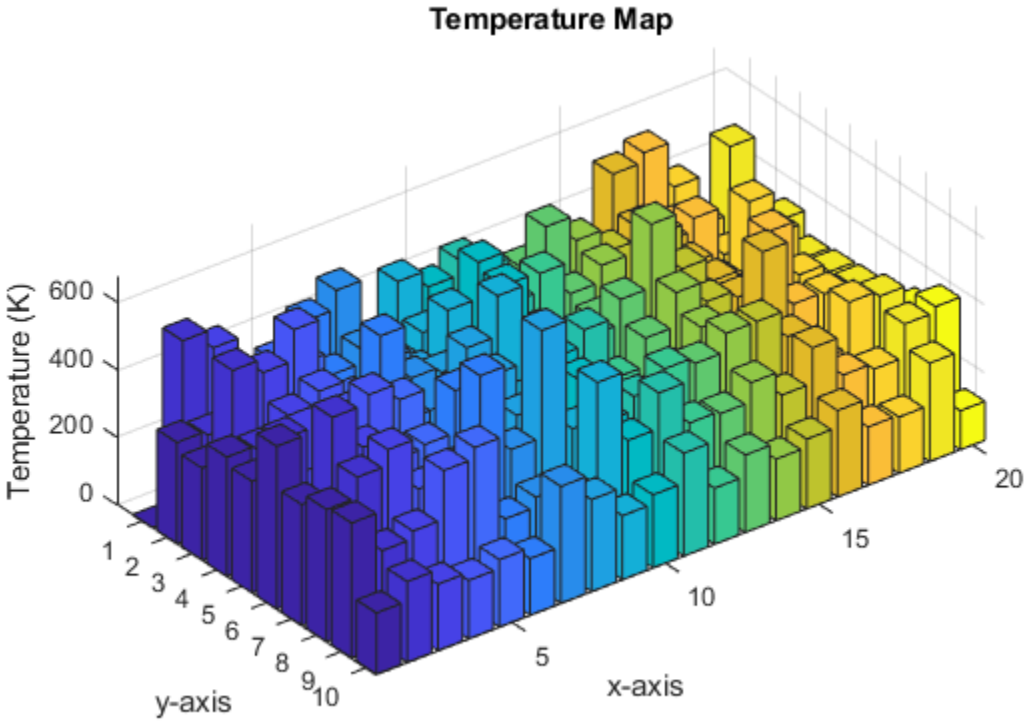

Voltage Over The Semiconductor Area
Voltage
0.1
0.08
0.06
0.04
0.02
0
100
50
0
Width
0
50
100
150
200
Length


Electric Field Quiver Across The Semiconductor Area
Width
100
90
80
70
60
50
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200
Length


2-D Particle pathing
×10^-7
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Width
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Length
×10^-7


Current Density vs Time
0.012
0.01
0.008
0.006
0.004
0.002
0
Current Density
0
0.2
0.4
0.6
0.8
1
1.2
Time
×10^-12


Electron Density Map
Number of electrons
80
60
40
20
0
10
×10^-8
5
0
y-axis
0
5
10
15
20
×10^-8
x-axis


Temperature Map
Temperature (K)
600
400
200
0
1
2
3
4
5
6
7
8
9
10
y-axis
5
10
15
20
x-axis

Part 2

Calculations Setting Constants

```
mElec = 9.11e-31; % Electron rest mass (kg)
mEff = 0.26*mElec; % Effective mass (kg)
kb = 1.381e-23; % Boltzmann's Constant (J/K)
T = 300; % Room Temperature (K)
L = 200e-9; % Length (m)
W = 100e-9; % Width (m)
q = 1.60218e-19; % Elemntary Charge (C)
Lb = 40e-9; % Box length
Wb = 40e-9; % Box width
s = 1e-9; % mesh spacing
nx = ceil(L/s + 1);
ny = ceil(W/s + 1);

cond = 1; % Conductive
res = 1e-2; % Resistive

% Generating the conductivity map
condMap = zeros(nx,ny);
for i = 1:nx
   for j = 1:ny
       if (i-1>0.5*(L-Lb)/s) && ((i-1)<0.5*(L+Lb)/s) && (((j-1)<Wb/s||
(j-1)>(W-Wb)/s))
           condMap(i,j) = res;
       else
           condMap(i,j) = cond;
       end
   end
end

G = sparse(nx*ny);
Bc = zeros(1,nx*ny);

for i = 1:nx
    for j = 1:ny
        n = j +(i-1)*ny; %Mapping Eq

        %Setting Boundary conditions
        if i == 1 || i == nx
            G(n,n) = 1;
            if i == 1
                Bc(n) = 1;
            end
        elseif j == 1 || j == ny
            nxm = j+(i-2)*ny;
            nxp = j+i*ny;
            nyp = j+1+(i-1)*ny;
            nym = j-1+(i-1)*ny;

            %Resistances from the conductivity map
```

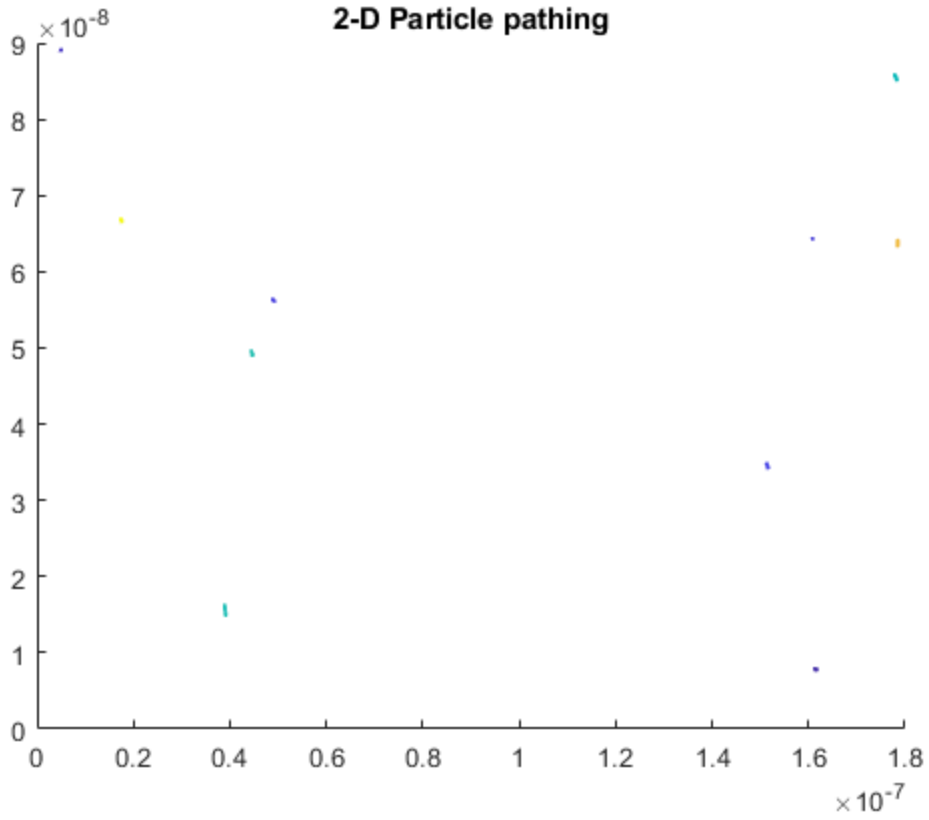
```
            rxm = (condMap(i,j)+condMap(i-1,j))/2;
            rxp = (condMap(i,j)+condMap(i+1,j))/2;

            if j == 1
                ryp = (condMap(i,j)+condMap(i,j+1))/2;
                G(n,n) = -(rxm+rxp+ryp);
                G(n,nyp) = ryp;
            else
                rym = (condMap(i,j)+condMap(i,j-1))/2;
                G(n,n) = -(rxm+rxp+rym);
                G(n,nym) = rym;
            end
            G(n,nxm) = rxm;
            G(n,nxp) = rxp;

        % Rest of the nodes
        else
            nxm = j+(i-2)*ny;
            nxp = j+i*ny;
            nym = j-1+(i-1)*ny;
            nyp = j+1+(i-1)*ny;

            % Getting the resistances from the conduction map
            rxm = (condMap(i,j)+condMap(i-1,j))/2;
            rxp = (condMap(i,j)+condMap(i+1,j))/2;
            ryp = (condMap(i,j)+condMap(i,j+1))/2;
            rym = (condMap(i,j)+condMap(i,j-1))/2;

            % Assigning the node equations
            G(n,n) = -(rxm+rxp+rym+ryp);
            G(n,nxm) = rxm;
            G(n,nxp) = rxp;
            G(n,nym) = rym;
            G(n,nyp) = ryp;
        end
    end
end

%solving and plotting
V = G\Bc';
mappedV = zeros(nx,ny);
for i = 1:nx
    for j = 1:ny
        n = j+(i-1)*ny;
        mappedV(i,j) = V(n);
    end
end

% a) Surface plot of Potential over the region

[xaxis, yaxis] = meshgrid(0:s:L,0:s:W);
figure(7)
surf(xaxis',yaxis',mappedV)
hold on
```

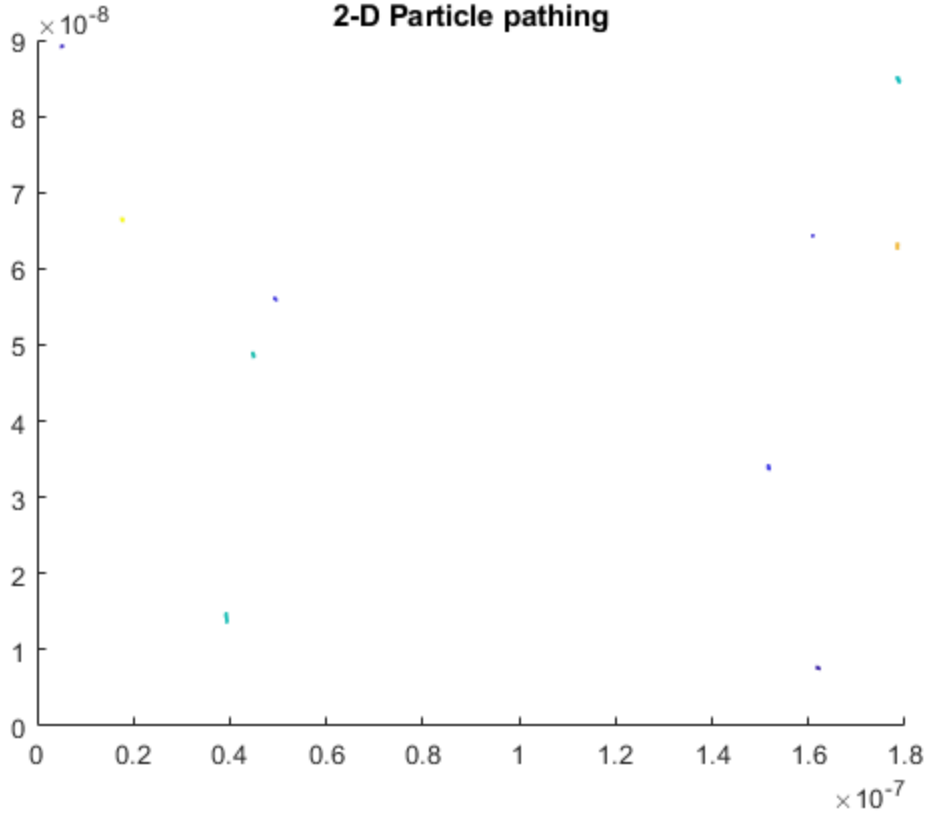
```
imagesc([0 L],[0 W],mappedV')
title('Potential Over the Region')
hold off

% b) 2-D Electric Field Vector Quiver

[Ex, Ey] = gradient(-mappedV',1e-9);

figure(8)
quiver(xaxis',yaxis',Ex',Ey')
xlim([0 L])
ylim([0 W])
title('E Field Quiver')

% c) 2-D Particle Pathing Plot
% Reusing the code from Q1 with Ex and Ey as inputs

% Finding the thermal velocity
Vth = sqrt((2*kb*T)/mEff); %Thermal Velocity in m/s
mt = 0.2e-12; % mean time (s)

% Modeling the electron paths

numPar = 10000; % Number of particles

% Assigning particle positions
posX = L.*rand(numPar,2);
posY = W.*rand(numPar,2);
% Reassigning electron position so none of them spawn inside the
 contacts
for i = 1:numPar
    while ((posX(i,2) > 80e-9 && posX(i,2) < 120e-9) && (posY(i,2) >
 60e-9 ||  posY(i,2) < 40e-9))
        posX(i,2) = L.*rand(1,1);
        posY(i,2) = W.*rand(1,1);
    end
end
posX(:,1) = posX(:,2);
posY(:,1) = posY(:,2);

spacialStep = sqrt(L^2+W^2)/1000;
stepTime = 0.5*spacialStep/Vth; %using a reduced steptime for this
 part

% Assigning each particle a random velocity (acceleration will be
 taken into account during the iteration)
Vx = randn(numPar,2)*sqrt((kb*T)/mEff);
Vy = randn(numPar,2)*sqrt((kb*T)/mEff);

% Displacement per step of each electron
dispX = stepTime*Vx(:,1);
dispY = stepTime*Vy(:,1);

colors = rand(numPar,3);
```

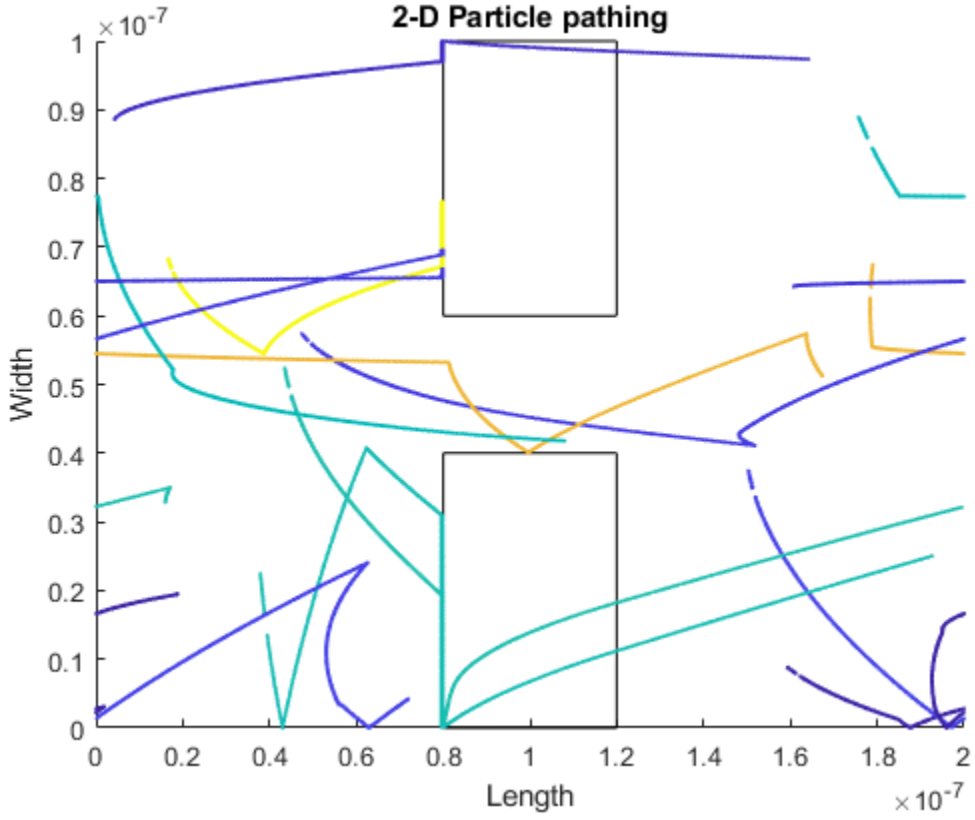
```
pScatter = 1 - exp(-stepTime/mt); % Probability of an electron
 scattering

% Positions before collision
collX = posX(:,1);
collY = posY(:,1);

ic = 0; % iteration counter for animated plot

for i = 1:1000
    for j = 1:numPar
        if((round(posX(j,1)*1e9) > 200) || (round(posY(j,1)*1e9)) >
 100)
            Fx = Ex(floor(posY(j,1)*1e9),floor(posX(j,1)*1e9)).*q;
            Fy = Ey(floor(posY(j,1)*1e9),floor(posX(j,1)*1e9)).*q;
        elseif((round(posY(j,1)*1e9)) <=0 || (round(posX(j,1)*1e9) <=
 0))
            Fx = Ex(ceil(posY(j,1)*1e9),ceil(posX(j,1)*1e9)).*q;
            Fy = Ey(ceil(posY(j,1)*1e9),ceil(posX(j,1)*1e9)).*q;
        else
            Fx = Ex(round(posY(j,1)*1e9),round(posX(j,1)*1e9)).*q;
            Fy = Ey(round(posY(j,1)*1e9),round(posX(j,1)*1e9)).*q;
        end
        ax = Fx/mEff; % Horizontal Acceleration
        ay = Fy/mEff; % Vertical Acceleration
        % Adding the extra velocity from the acceleration (x only)
        Vx(j,1) = Vx(j,1) + ax*stepTime;
        Vy(j,1) = Vy(j,1) + ay*stepTime;

        if(rand < pScatter)
            % generating new velocity/displacment values
            Vx(j,:) = randn(1)*sqrt((kb*T)/mEff) + ax*stepTime;
            Vy(j,:) = randn(1)*sqrt((kb*T)/mEff) + ay*stepTime;
            dispX(j,:) = stepTime*Vx(j,1);
            dispY(j,:) = stepTime*Vy(j,1);
            collX = posX(:,1);
            collY = posY(:,1);
        end

        if (posX(j,1)+dispX(j) > L)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)-L;
        elseif (posX(j,1)+dispX(j) < 0)
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j)+L;
        else
            dispX(j,:) = stepTime*Vx(j,1);
            posX(j,2) = posX(j,1)+dispX(j);
        end

        % Adding extra boundary conditions for particle collision with
 the
        % contacts
```

```
        % Top or bottom of the region
        if ((posY(j,1)+dispY(j) > W)|| (posY(j,1)+dispY(j) < 0))
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        % Approaching a contact from the right
        elseif ((posX(j,1) < 80e-9) && (posX(j,1) + dispX(j) > 80e-9)
 && ((posY(j,1) > 60e-9) || (posY(j,1) < 40e-9)))
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1)+dispX(j);
        % Approaching a contact from the left
        elseif ((posX(j,1)) > 120e-9 && (posX(j,1) + dispX(j) <
 120e-9) && ((posY(j,1) > 60e-9) || (posY(j,1) < 40e-9)))
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1)+dispX(j);
        % Top and bottom
        elseif (((posX(j,1) > 80e-9) && (posX(j,1) < 120e-9)) &&
 ((posY(j,1) > 40e-9) && (posY(j,1) < 60e-9)) && ((posY(j,1) +
 dispY(j) < 40e-9) || (posY(j,1) + dispY(j) > 60e-9)))
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        % Left Corner
        elseif ((posX(j,1) + dispX(j) == 80e-9) && ((posY(j,1) +
 dispY(j) == 40e-9) || (posY(j,1) + dispY(j) == 60e-9)))
            dispY(j) = -dispY(j);
            dispX(j) = -dispX(j);
            posY(j,2) = posY(j,1)+dispY(j);
            posX(j,2) = posX(j,1)+dispX(j);
        else
            posY(j,2) = posY(j,1)+dispY(j);
        end
    end
    if (ic == 0)
       figure(9);
       scatter(posX(1:10, 2),posY(1:10,2),1,colors(1:10,1));
       hold on;
       plot([80e-9,80e-9],[0,40e-9],'k',[80e-9,120e-9],
[40e-9,40e-9],'k',[120e-9,120e-9],[40e-9,0],'k',[80e-9,120e-9],
[0,0],'k')
       plot([80e-9,80e-9],[100e-9,60e-9],'k',[80e-9,120e-9],
[60e-9,60e-9],'k',[120e-9,120e-9],[60e-9,100e-9],'k',[80e-9,120e-9],
[100e-9,100e-9],'k')
       title('2-D Particle pathing');
       xlabel('Length');
       ylabel('Width');
       xlim([0 L]);
       ylim([0 W]);
    elseif (ic < 1000)
       title('2-D Particle pathing');
       scatter(posX(1:10, 2),posY(1:10,2),1,colors(1:10,1));
       hold on;
    else
       title('2-D Particle pathing');
       scatter(posX(1:10, 2),posY(1:10,2),1,colors(1:10,1));
```

```
        hold off;
    end
    pause(0.001);
    posX(:,1) = posX(:,2);
    posY(:,1) = posY(:,2);
    ic = ic + 1;
end
% The code works as intended for the most part, however there is
 sometimes
% an index out of bounds error when calculating Fx and Fy, the mesh
 spacing
% from parts a and b (assignment 2) needs to be manually adapted to
 fit the
% number of electrons and iterations that need to be done for this
 part.
% The number of electrons is reduced in the graph due to me not having
% enough RAM and/or computing power to simulate 1000 electrons for
 1000 steps.
```




2-D Particle pathing
×10^-8
9
8
7
6
5
4
3
2
1
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
×10^-7

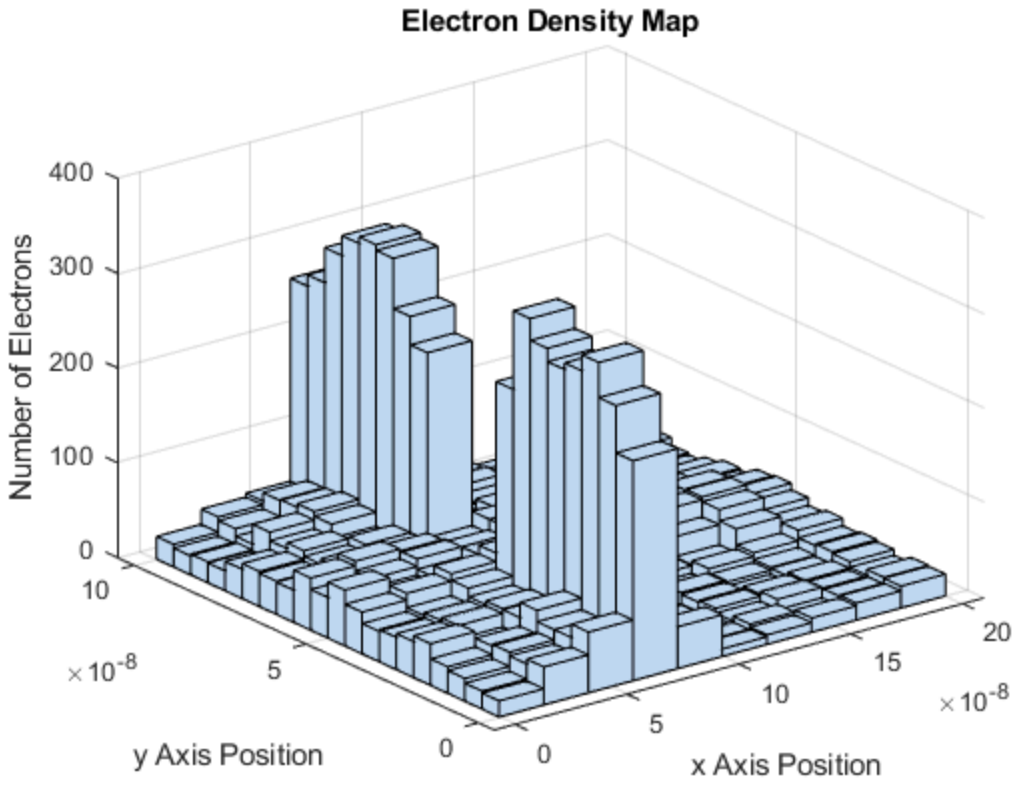

2-D Particle pathing
×10^-7
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Width
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Length
×10^-7

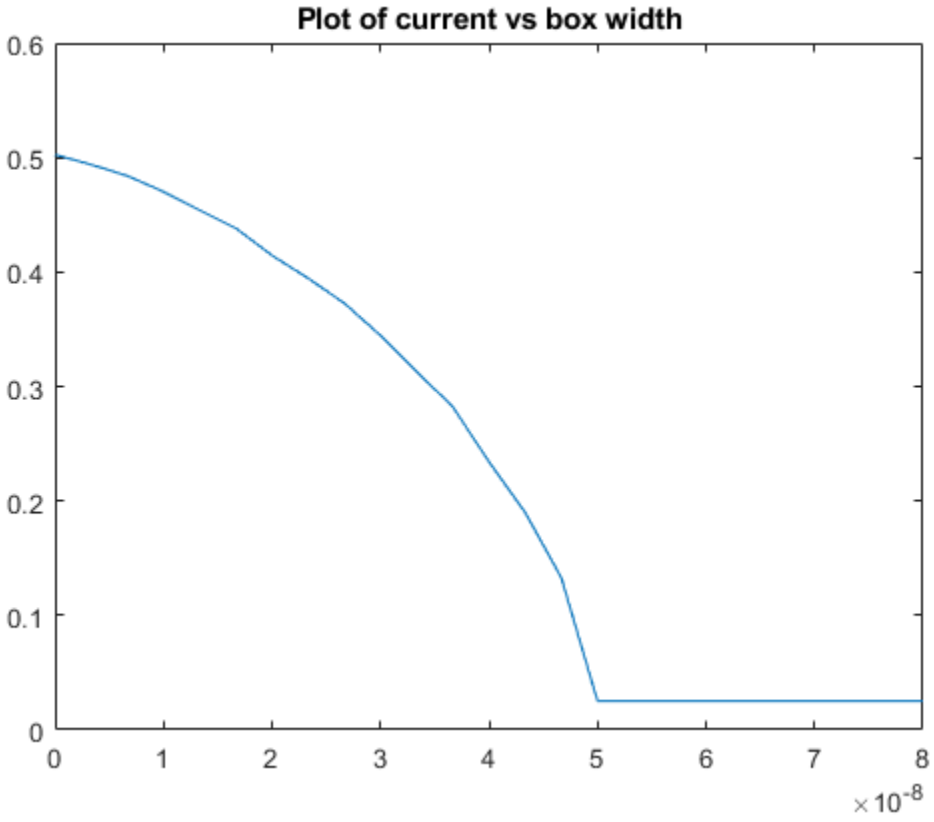
Part 3

```
% a) Electron Density map and comments
figure(10)
hist3([posX(:,1),posY(:,1)],[10,20])
title('Electron Density Map')
xlabel('x Axis Position')
ylabel('y Axis Position')
zlabel('Number of Electrons')
% The electrons seemed to mostly accumulate around the contacts
 towards the
% center, however this may be due to a bug in the code that allows
 some
% electrons to slip into te boxes through the top and bottom of the
 region.

% b) Plotting the current vs bottleneck width
% To plot the current vs bottleneck/box width we would just need to
 reuse the code from Assignment 2, but with the parameters used in
 this case
Wb = linspace(0,80e-9,25);
for i = 1:25
    I(i) = Curr(1e-9, 0.01, Wb(i));
end
figure(11)
plot(Wb,I)
title('Plot of current vs box width')

% Creating a reuseable function that outputs the current
% This will be used for the remaining plots

function [I] = Curr(s, cond, Wb)
W = 100e-9;
L = 200e-9;
Lb = 40e-9;
nx = floor(L/s+1);
ny = floor(W/s+1);

    condMap = zeros(nx,ny);
    for i = 1:nx
        for j = 1:ny
            if (i-1>0.5*(L-Lb)/s) && ((i-1)<0.5*(L+Lb)/s) &&
 (((j-1)<Wb/s||(j-1)>(W-Wb)/s))
                condMap(i,j) = cond;
            else
                condMap(i,j) = 1;
            end
        end
    end

    G = sparse(nx*ny);
    Bc = zeros(1,nx*ny);
```

```
for i = 1:nx
    for j = 1:ny
        n = j +(i-1)*ny; %Mapping Eq
        %Setting Boundary conditions
        if i == 1 || i == nx
            G(n,n) = 1;
            if i == 1
                Bc(n) = 1;
            end
        elseif j == 1 || j == ny
            nxm = j+(i-2)*ny;
            nxp = j+i*ny;
            nyp = j+1+(i-1)*ny;
            nym = j-1+(i-1)*ny;

            % Resistances from the conductivity map
            rxm = (condMap(i,j)+condMap(i-1,j))/2;
            rxp = (condMap(i,j)+condMap(i+1,j))/2;

            if j == 1
                ryp = (condMap(i,j)+condMap(i,j+1))/2;
                G(n,n) = -(rxm+rxp+ryp);
                G(n,nyp) = ryp;
            else
                rym = (condMap(i,j)+condMap(i,j-1))/2;
                G(n,n) = -(rxm+rxp+rym);
                G(n,nym) = rym;
            end

            G(n,nxm) = rxm;
            G(n,nxp) = rxp;

            % Rest of the nodes
            else
                nxm = j+(i-2)*ny;
                nxp = j+i*ny;
                nym = j-1+(i-1)*ny;
                nyp = j+1+(i-1)*ny;

                % Getting the resistances from the conduction map
                rxm = (condMap(i,j)+condMap(i-1,j))/2;
                rxp = (condMap(i,j)+condMap(i+1,j))/2;
                ryp = (condMap(i,j)+condMap(i,j+1))/2;
                rym = (condMap(i,j)+condMap(i,j-1))/2;

                % Assigning the node equations
                G(n,n) = -(rxm+rxp+rym+ryp);
                G(n,nxm) = rxm;
                G(n,nxp) = rxp;
                G(n,nym) = rym;
                G(n,nyp) = ryp;
            end
    end
end
```

```
    V = G\Bc';
    mappedV = zeros(nx,ny);
    for i = 1:nx
        for j = 1:ny
            n = j+(i-1)*ny;
            mappedV(i,j) = V(n);
        end
    end

    [Ey, Ex] = gradient(-mappedV);
    Jx = condMap.*Ex;
    Jy = condMap.*Ey;

    I = sum(Jx(1,:));
end

% c) The main issue with this simulation is its lack of efficiency,
 the
% code takes too long to run and takes up too much memory; it needs to
 be
% optimized. With more efficient code we can simulate more data points
 much
% faster, and given more data we can get more accurate results. The
 second
% issue with the coding is that it is buggy and imperfect, there are
 cases
% where electrons leak into the boxes through the bottom and top of
 the
% region. I have not yet been able to pinpoint the source of this
 issue. In
% conclusion, the code preforms the simulation well and is a huge
% improvement over the code from assignment 1 which was much more
 faulty.
% If time had permitted, I would probably have focused on debugging
 and
% optimization.
```


Electron Density Map
Number of Electrons
400
300
200
100
0
10
5
0
×10^-8
y Axis Position
0
5
10
15
20
×10^-8
x Axis Position


Plot of current vs box width
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7
8
×10^-8

Published with MATLAB® R2020b