ELEC 4700

Assignment - 1

Monte-Carlo Modeling of Electron Transport

Due: Sunday, Feb. 2, 2020 23:59

Wasam Al-bayti

101041465

Date Submitted: Feb. 2, 2020

1.0 Introduction

This lab deals with electron motion modeling in a semiconductor area. The semiconductor device has specific vertical and horizontal measurements. Monte-Carlo modeling technique will be used to model the motion of electrons in MATLAB. The first section will be a simple random electron modeling, the second section will deal with collisions and the third section will include a new boundary, "bottle neck" within the semiconductor device.

2.0 Electron Modeling

A simplistic Monte-Carlo model was implemented to model the electrons in the silicon as particles. The thermal velocity was calculated using the equation below:

$$v_{th} = \sqrt{\frac{2KT}{m_n}}$$

Using the below parameters, the thermal velocity was calculated in MATLAB.

```
m = 0.26*9.11e-31; % Effective mass of electron
num_particles = 10; % Initialize the number of particles within the region
T =300; % Temp in kelvin
k = 1.3806e-23; % Boltzmann constant

% thermal velocity vth
vth = sqrt(2*k*T/m)
```

The calculated thermal velocity was 1.8701e+05 m/s.

The mean time between collision was calculated in MATLAB with this equation:

```
% mean free path
t = 0.2e-12 * vth
```

The calculated mean free path is 3.7402e-08 m.

In order to model Electron movement, the position of particles was randomly assigned with a constant speed in a random direction.

```
% Assigning each particle, a random location in the x-y plane within the
region
inital_xPosition = length * rand(num_particles,1);
inital_yPosition = width * rand(num_particles,1);

% Assign each particle with fixed velocity given by vth
angle = 2*pi*rand(num_particles,1);
inital_xVelocity = ones(num_particles,1);
inital_yVelocity = ones(num_particles,1);
```

Next, Newton's law of motion was used to update the particle location. The simulated code is shown below:

```
%simulate
for i=1:1000

    % Updating particle location using Newton's laws of motion
    inital_xPosition = inital_xPosition + inital_xVelocity.*delta_T;
    inital_yPosition = inital_yPosition + inital_yVelocity.*delta_T;

    % boundary condition where the particle reflects at the same angle in
    inital_xPosition(inital_xPosition<=0) =
inital_xPosition(inital_xPosition<=0)+1;
    inital_xPosition(inital_xPosition>=1) =
inital_xPosition(inital_xPosition>=1)-1;

    % Periodic boundary condition where the particle jumps to the opposite
edge
    inital_yVelocity(inital_yPosition>=width)= -
inital_yVelocity(inital_yPosition>=width);
    inital_yVelocity(inital_yPosition<=0) = -
inital_yVelocity(inital_yPosition<=0);

    % Calculating the average temperature
    temp =
m*mean(sqrt((sum(abs(inital_xVelocity))/num_particles)^2+(sum(abs(inital_yVel
ocity))/num_particles)^2).^2)/k;


    %Plot the movement of the particles
    subplot(3,1,1)
    plot(inital_xPosition,inital_yPosition,'.')
    title('Particle trajectories')
    xlabel('X position')
    ylabel('Y position')
    xlim([0 length])
    ylim([0 width])
    hold on
    pause(.01)

    % Temperature plot
    subplot(3,1,2)
    plot(i*delta_T,temp,'o')
    title('Temperature plot')
    ylabel('Temperature (k)')
    xlabel('Time (s)')
    hold on
end
```

The particle trajectories plot is shown below. Only 20 particles were plotted, this can easily be increased by changing the variable, num_particles. With this low number of particles plotted, the behavior of the particles is visible and observable. It is seen that when hitting the top axis or bottom axis, the particle reflects at an equal angle. When the particle goes past the left or right boundary it will appear in the

opposite boundary. The temperature is seen to be constant throughout the movement of the particles. This won't be the case in the next part when the collision of the particles will be coded in.

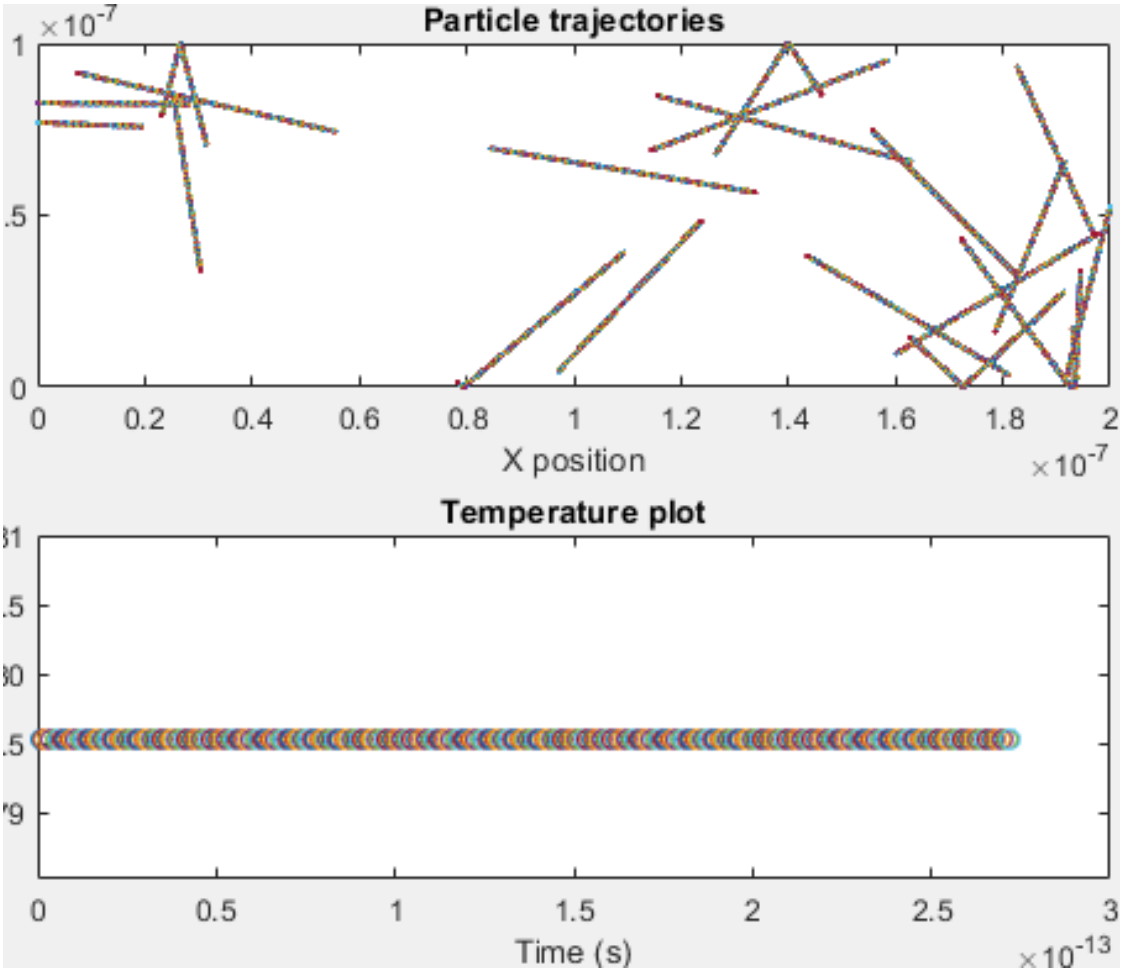

Figure 1-2/ trajectory plot and temperature plot

3.0 Collisions with Mean Free Path

The scattered electrons were modeled using the code below:

```
%Scatter electrons
    if (prob > rand())
        inital_xVelocity=randn(num_particles,1).*vth/sqrt(2);
        inital_yVelocity=randn(num_particles,1).*vth/sqrt(2);
        velAvg = sqrt(inital_xVelocity.^2 + inital_yVelocity.^2);

        collisionsNum=collisionsNum+1;
        diff=i-count;
        count=i;
        total=total+(diff*delta_T);
        meanVelocity=meanVelocity+(mean(velAvg));

        %Running Mean Free Path
        averageMFP=(total/collisionsNum)*(meanVelocity/collisionsNum)
```

```
%mean time between collisons
meanCollisons = total/collisionsNum;
```

The plot is shown below of the scattered particles. It is seen that the particles don't necessarily move in a straight direction as before, but now, the particles could randomly take a different direction whilst moving on its' own.

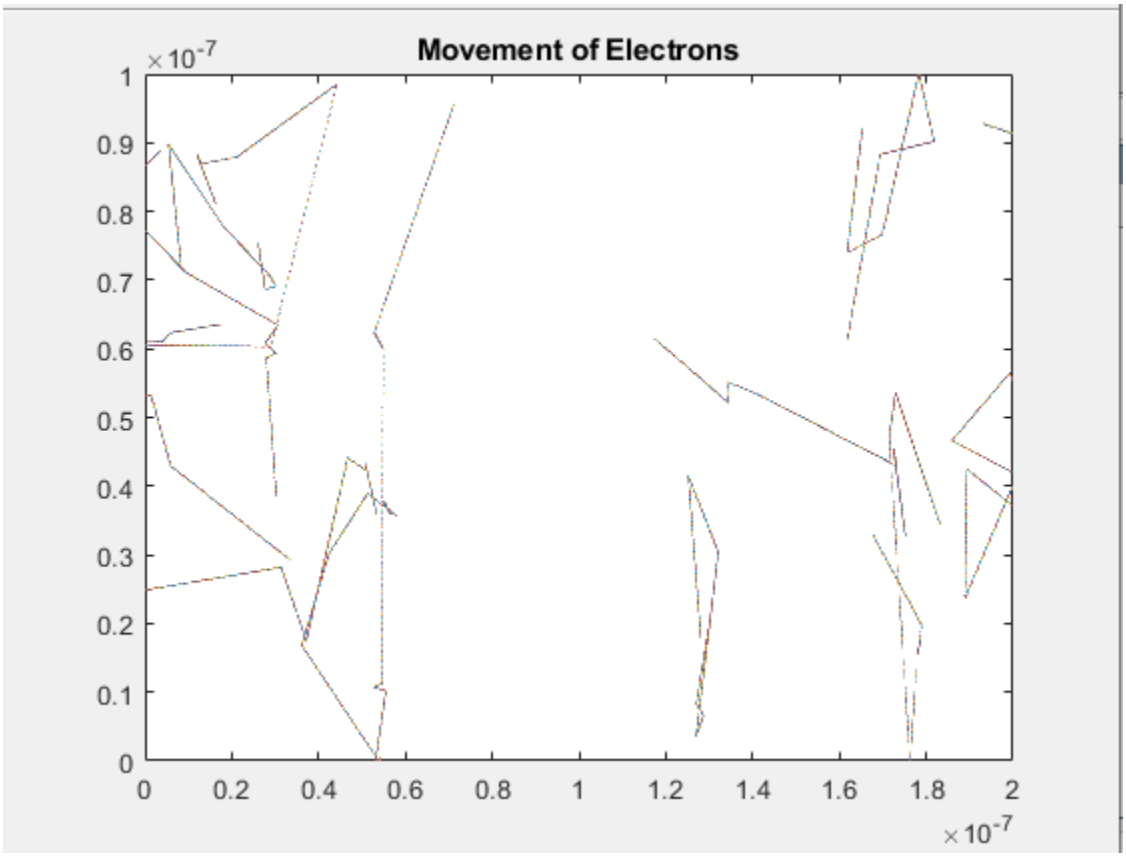

Figure 3: Movement of electrons with scatter

The plot below shows the temperature of the particles. Unlike last time, the temperature is not constant. But after letting the particles move for a while, the average temperature can be taken in code by MATLAB, and it is observed that it is around 300 K.

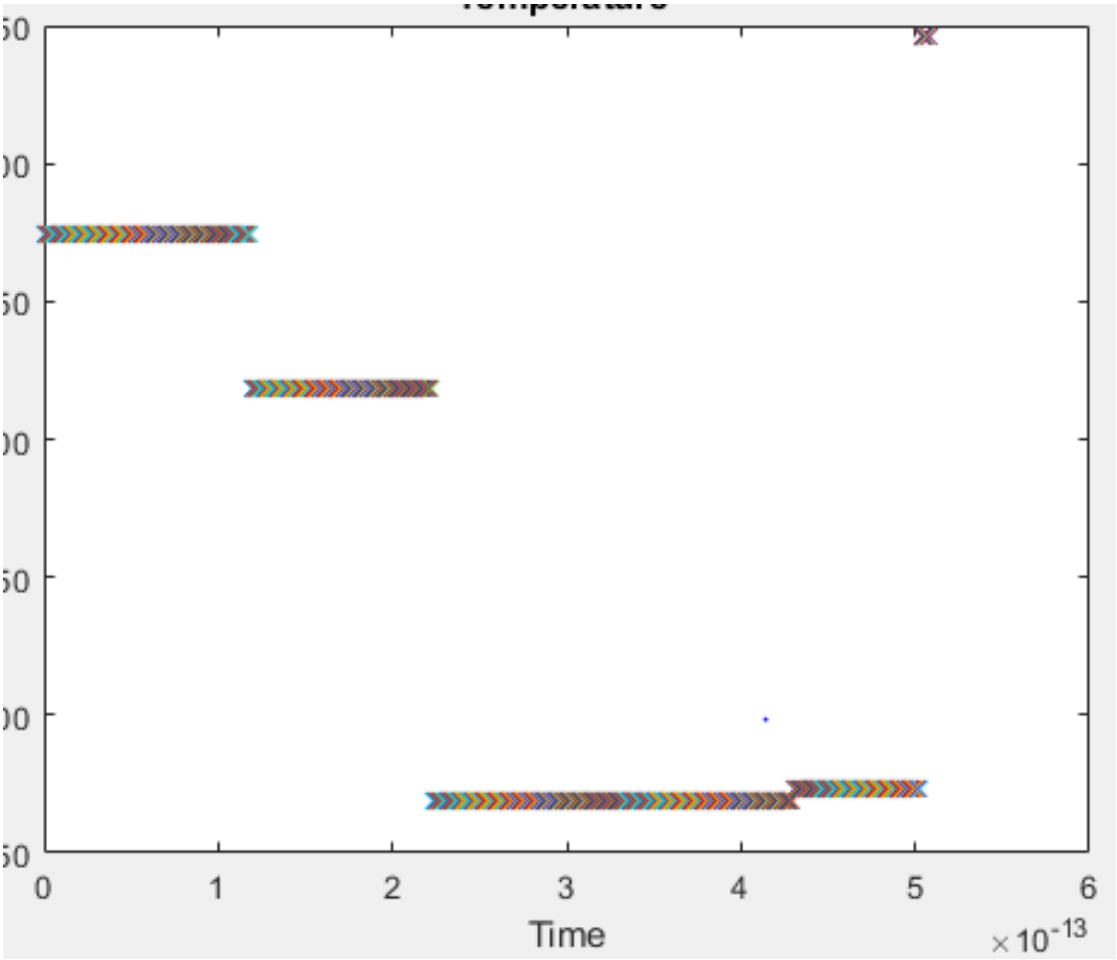

Figure 4: Temperature plot with collisions and scatter

The plot below shows the histogram of initial speeds and the number of particles that had a certain speed. The MATLAB code for plotting the histogram is shown below:

```
histogram(velAvg,5)
title('Electron Speed')
xlabel('Speed (m/s)')
ylabel('count')
hold on
```

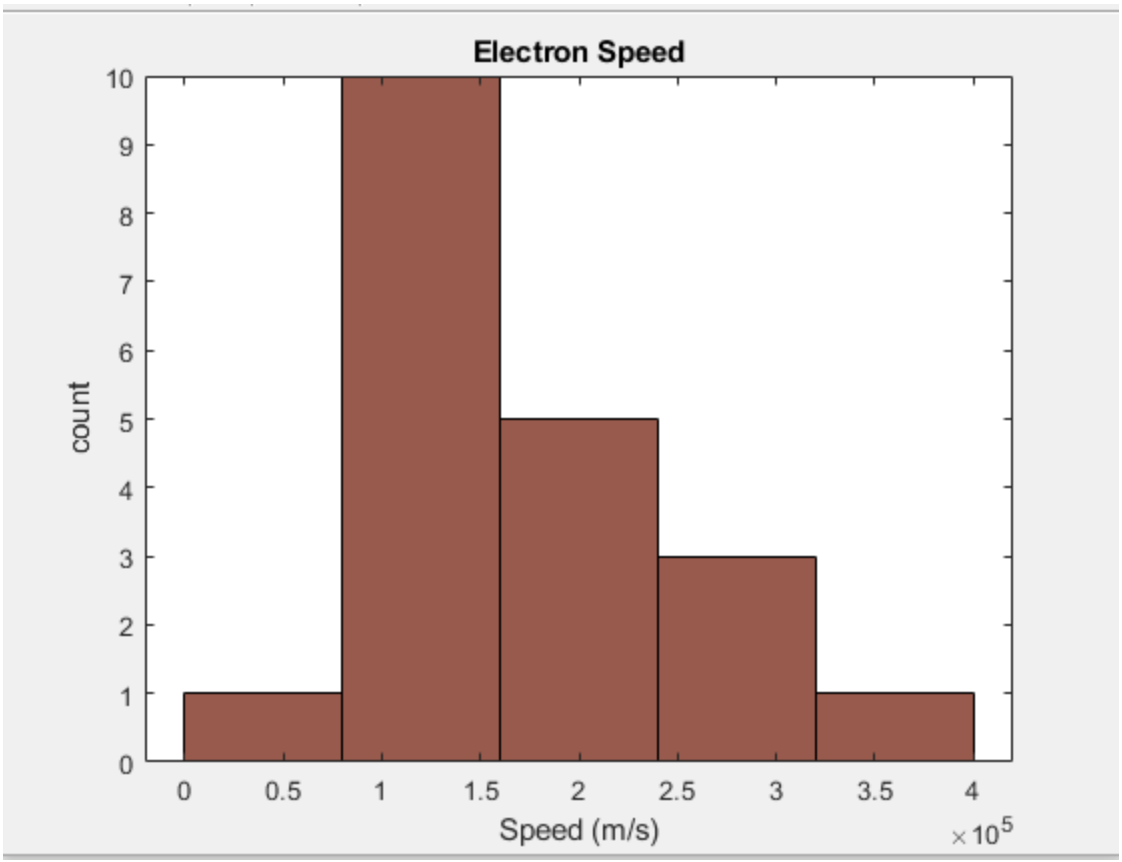

Figure 5: Electron speed histogram

The actual Mean Free Path can be calculated with the code below:

```
%Running Mean Free Path
averageMFP=(total/collisionsNum)*(meanVelocity/collisionsNum)
```

it was calculated to be 3.2474e-08 which is close to the theoretical in the first part.

The mean time between collisions was calculated with the code:

```
%mean time between collisions
meanCollisons = total/collisionsNum
```

it was calculated to be 1.6760e-13 seconds.

4.0 Enhancements

For this part, a bottle neck boundary is added to the inner rectangle.

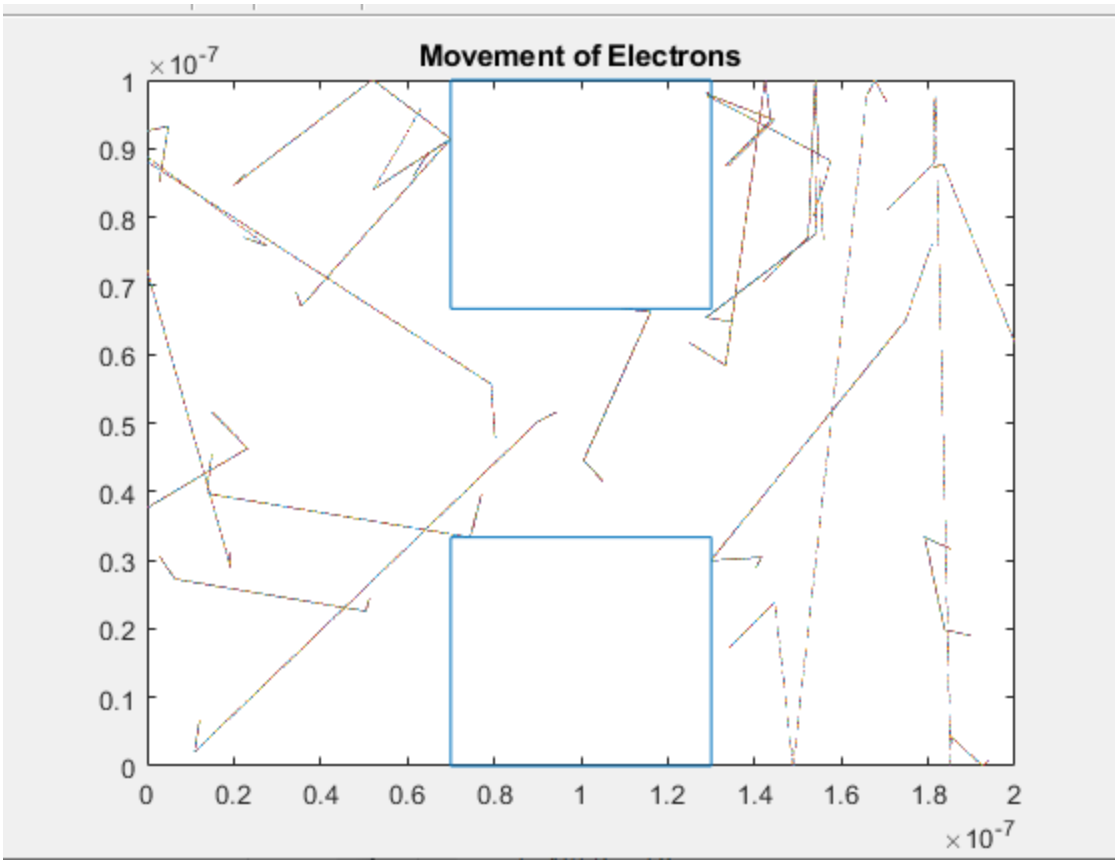

Figure 6: Bottle neck electron trajectory

The plot above shows the bottle neck where the particles are generated outside of it and when the particles encounter it, the particles reflect off and do not cross it. Again, this is only 20 particles, more can be generated.

The bottleneck was generated using the parameters below:

```
% Setting up bottleneck
  line([0.7*length/2 0.7*length/2],[w 2*width/3])
  line([1.3*length/2 1.3*length/2],[w 2*width/3])
  line([0.7*length/2 1.3*length/2],[width width])
  line([0.7*length/2 1.3*length/2],[2*width/3 2*width/3])
  line([0.7*length/2 0.7*length/2],[0 width/3])
  line([1.3*length/2 1.3*length/2],[0 width/3])
  line([0.7*length/2 1.3*length/2],[0 0])
  line([0.7*length/2 1.3*length/2],[width/3 width/3])
  hold on
  pause(0.01)
```

The electron density map and the final temperature density map is shown below. For both plots, surf function was used. For the electron density map, it is seen that the electrons final position is around the bottleneck but not on it. For the temperature density map, it shows a constant temperature. This might be incorrect since temperature is changing due to collisions. The code to get both these plots is shown below:

```
[xgradient,ygradient] = meshgrid(0:(length/10):length, 0:(width/10):width);

%This array will count the number of particles in each division of the frame
electronV=zeros(10,10);

temperatureV=zeros(10,10);
electrons_num=0;
velocityTot=0;

for i=1:10
    min_x=xgradient(1,i);
    max_x=xgradient(1,i+1);
    for k =1:10
        min_y=ygradient(k,1);
        max_y=ygradient(k+1,1);

        for j=1:num_particles
            %Check to see if particle is within this division of the frame
            if( (inital_yPosition(j)>min_y & inital_yPosition(j)<max_y) &
(inital_xPosition(j)<max_x & inital_xPosition(j)>min_x))

velocityTot=velocityTot+sqrt(inital_xVelocity(j)^2+inital_yVelocity(j)^2);
                electrons_num=electrons_num+1;
                electronV(i,k)=electronV(i,k)+1;

            end
        end
    end
end
```

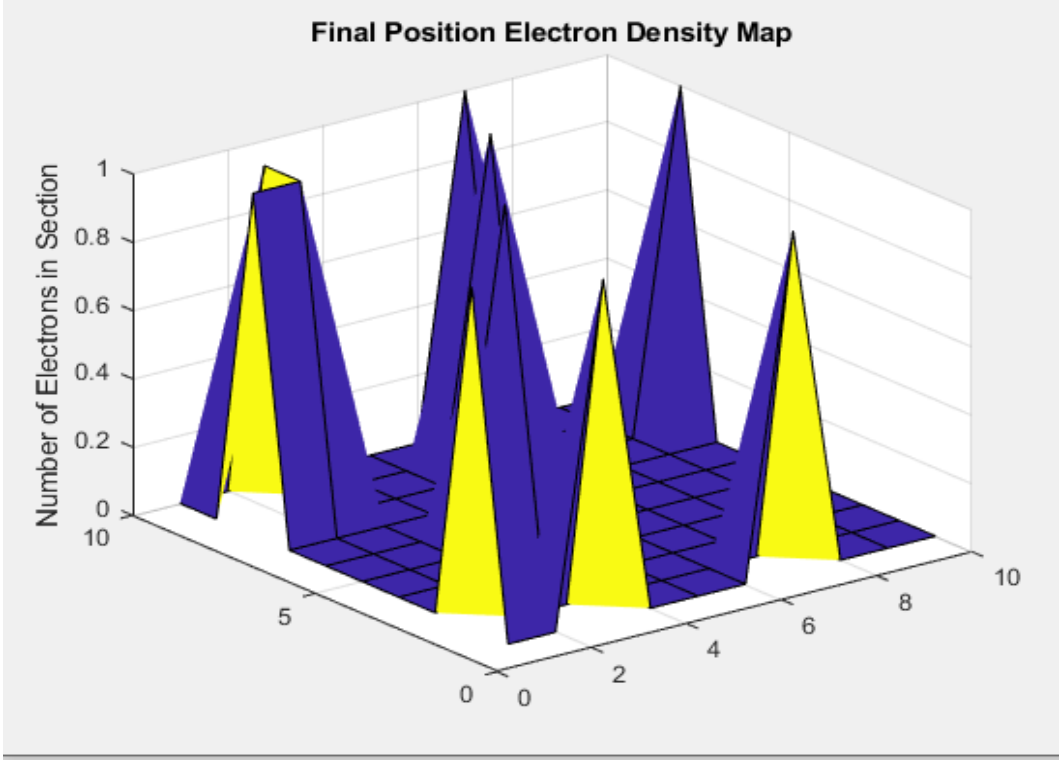

Figure 7: Electron density map

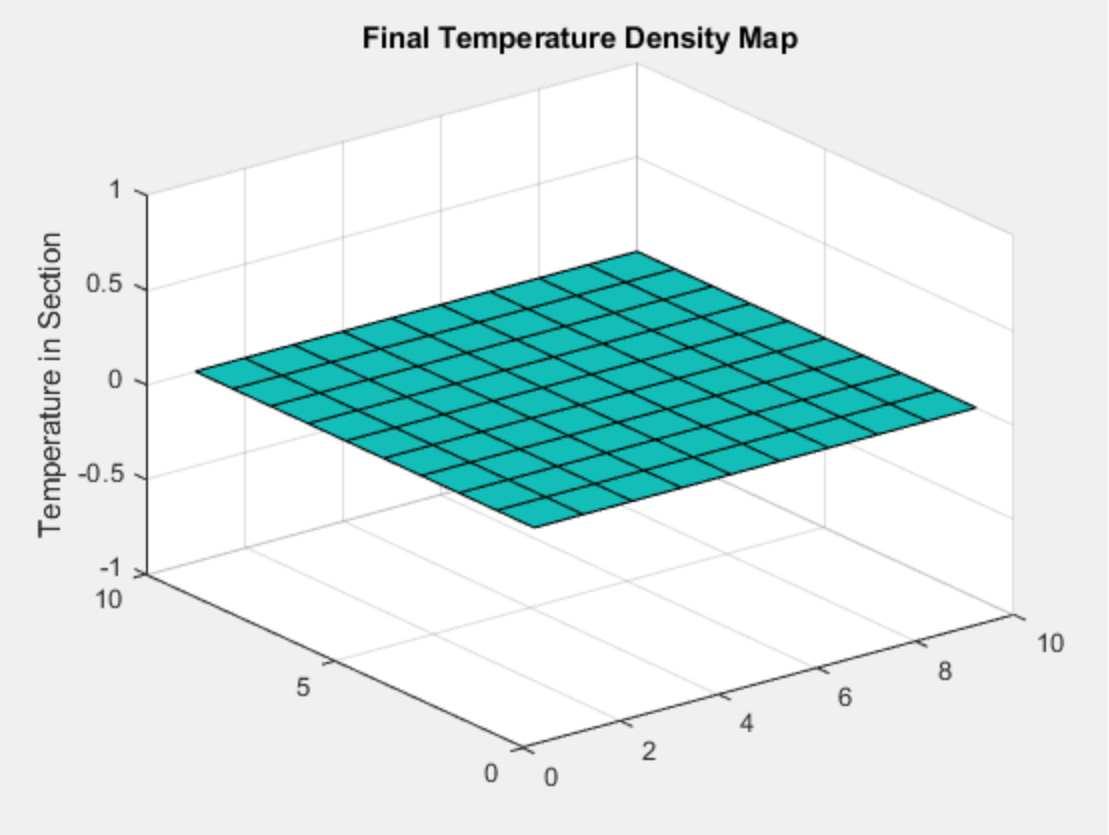

Figure 8: Temperature density map

The temperature density map above is incorrect, there should be some change in the temperature.

5.0 Conclusion

In conclusion, Monte-Carlo modeling was used to model electron motion within a semiconductor device. All the results came as expected except for the temperature density map. It showed a flat surface, meaning the temperature does not change when in fact, in figure 4, we can see that the temperature varies due to collisions. This incorrect result is from a mistake within the code which could be fixed.