Assignment 1-
Monte- Carlo Modeling of Electron Transport

ELEC 4700

Neale Skinner

Due 2020-02-02

Table of Contents

Introduction:

In this assignment we are modeling electron movement, through the Monte- Carlo method, while in a semiconductor device. This experiment is conducted entirely through Matlab for accurate plotting of movement, temperature, velocity and density. The material of choice will have length of 200nm and a height of 100nm. We will initially just be looking at how electrons will react in that limited space with reflective walls but will move onto seeing the further reaction when more limitations are placed inside the area. In this assignment the probability of scatter will also be modeled along with how this will affect our velocities when we use a Maxwell distribution to scatter. Lastly, we will be plotting the density and temperature per unit nm with a 3D representation. it is to be noted that for this assignment we are given a list of common variables that will be used throughout with initial values, they can be seen below.

```
T=300;                  %Temp in K
K=1.38e-23;             %Boltsmann constant
Tmn=0.2e-12;            %mean time between collisions
Mo=9.11e-31;            %rest mass
Mn=0.26*Mo;             %effective mass of electrons
L=200e-09;              %Length of region
W=100e-09;              %Width of region
PopE=25;                %number of particles
Vth=sqrt((K*T)/(Mn));   %Thermal velocity
TimeS=15e-15;           %time step of 15ns
iterations=100;
```

Part 1: Electron Modeling

In the initial stages of the assignment we are just looking at the specular reflection of electron in a closed space. Each electron is randomly assigned a starting point, inside the closed area, and a random velocity. To do this we will be using the code below;

```
Vth=sqrt((K*T)/(Mn));   %Thermal velocity
Angle = rand(PopE,1)*2*pi;   %random angle
Pos = [rand(PopE,1)*L rand(PopE,1)*W Vth*cos(Angle) Vth*sin(Angle)];
```

After this code is run, we will be given a matrix that is the value of PopE, in this case 25, by 4 elements wide. PopE holds the population of electrons we are choosing to use; I have written the code this way so that PopE is a global variable and changing its value once will change it throughout the assignment. In order to follow the path of the electron we will need to loop through some code for the number of steps we choose to see.

```
initialX=Pos(:,1);     %array of the initial X values in Position array
    initialY=Pos(:,2); %array of the initial Y values in Position array
    colorstring=rand(PopE,1);           %array of values corresponding
to shades of color, will be used in plotting
for i= 1 : iterations
    NextX=initialX + Pos(:,3) * TimeS;      %the next X position will
be the initial plus a time step of velocity in X direction
    NextY=initialY + Pos(:,4) * TimeS;      %the next Y position will
be the initial a plus a time step of velocity in Y.
```

```
    OverX=NextX > L;   %check for values in next position that are
greater than bounds
    NextX(OverX)=NextX(OverX)-L;   %subtract from the bound to reflect
    UnderX=NextX < 0;              %check for values less than bounds
    NextX(UnderX)= NextX(UnderX) + L;   %reflect bound
    OverY=NextY > W;              %check for values greater then bounds
    NextY(OverY)= 2*W -NextY(OverY) ; %subtract from bounds to reflect
    Pos(OverY,4)=- Pos(OverY,4);        %reverse the velocity
    UnderY= NextY < 0;                  %check for under bounds
    NextY(UnderY)=- NextY(UnderY);      %reflect
    Pos(UnderY,4)=- Pos(UnderY,4);      %reverse the velocity

    figure(1)
    subplot(2,1,1);
    scatter(initialX,initialY,PopE, colorstring,'.' );      %plot the
coordinates of the electron before the move
    hold on
    scatter(NextX, NextY,PopE, colorstring,'.' );           %plot the
coordinates of the electron after the move

    axis([0 L 0 W]);
    title (['Trajectories with MFP ', num2str(MFP)]);

    initialX=NextX;
    initialY=NextY;
end
```

the above code is designed to do just this. The code will be stepping through the electron movement by seeing where the velocity is taking the electron and time stepping slowly through the path. The plotting occurs inside the same loop so that we see in real time the movement of electrons. The output from the code is seen in figure 1.

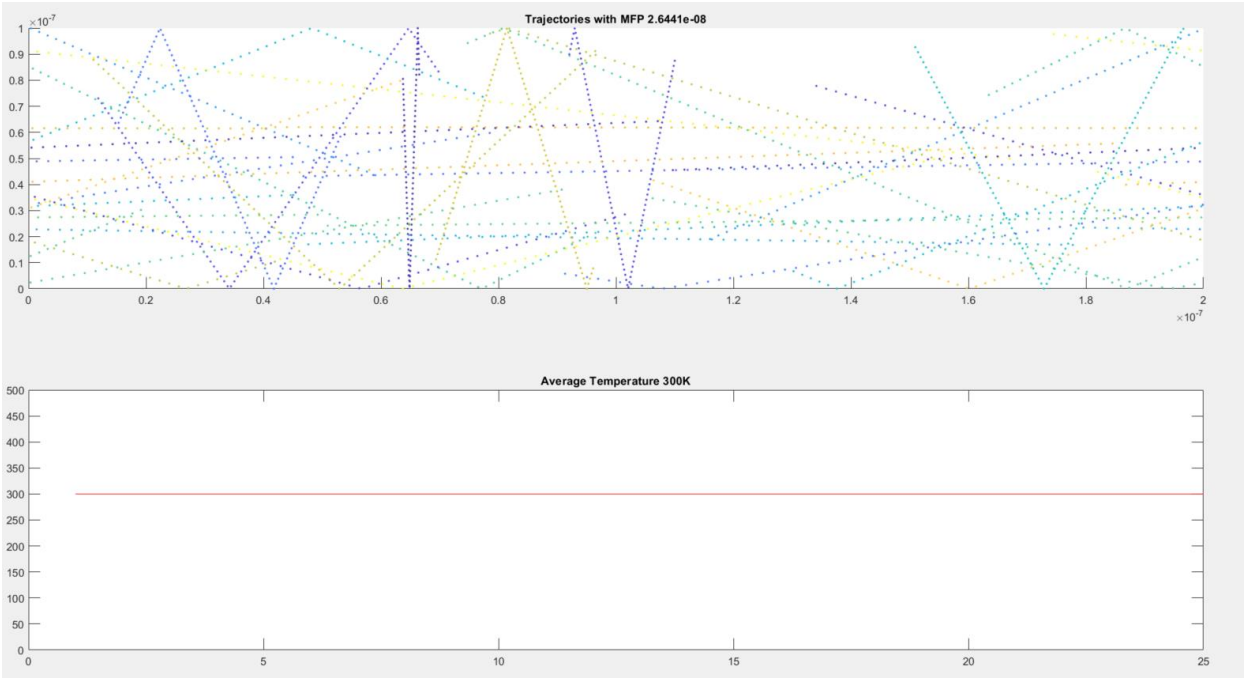

Figure 1: Electron Trajectory and Temperature of Semiconductor

Part 2: Collisions with Mean Free Path (MFP)

For this section of the assignment we will be using the same theory from part 1 but with a few enhancements. The first change is that we will be using a Maxwell- Boltzmann distribution for the velocities. We will also be checking to see if the particles will be scattering while moving while also keeping track of temperature over time and the Mean Free Path with mean time between collisions.

As mentioned before, this part will copy a lot from part 1 with a few enhancements. The first code added will be to define a variable that is the probability of scatter occurring. For this we will be using the probability of scatter code;

$$Pscat = 1 - e^{-\frac{dt}{Tmn}} \qquad (1)$$

In order to check each electron against this probability, we will create an array that is the same size as our Position arrays. Each corresponding cell will represent the electron (cell 1 in Position array and cell 1 in random array are electron 1) therefor, when an electron has a value greater than the probability of scatter, we will randomize that electrons velocities. In doing this the path will show a scatter of trajectory.

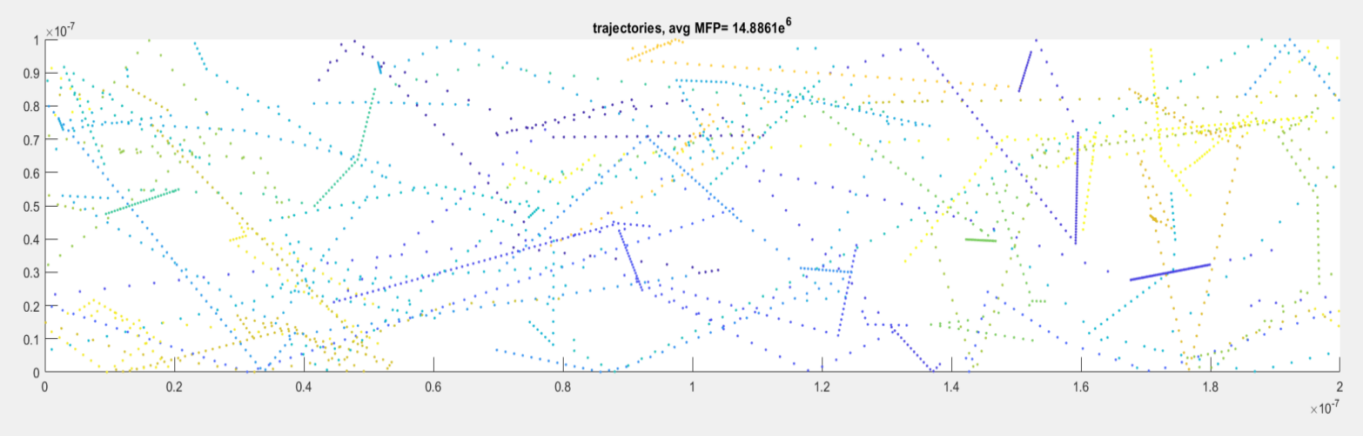

Figure 2: Trajectories with a probability of Scattering.

To accomplish this I have added 2 lines of code inside the loop seen in part 1.

```
OverV = rand(PopE,1) < Pscat ;        %check for probability of scatter
Pos(OverV,3:4)= random(VprobS,[sum(OverV),2]);
```

To accomplish the temperature over time we rearrange the thermal Velocity equation, seen below in equation 2, and solve to temperature.

$$Vth = \sqrt{\frac{KT}{M}} \qquad (2)$$

Where K is Boltzmann constant, T is temperature and M is the effective mass of electrons. The code for this segment is;

```
NextV=sqrt(sum(Pos(:,3).^2)/PopE + sum(Pos(:,4).^2)/PopE);  %save the
new average Velocity for use in Temp plot
NextT= T + ((Mn * (NextV.^2) )/K/PopE/2) ;
```

This code is placed inside the loop so that the temperature is able to be seen for the moving velocities from the probability of scattering.

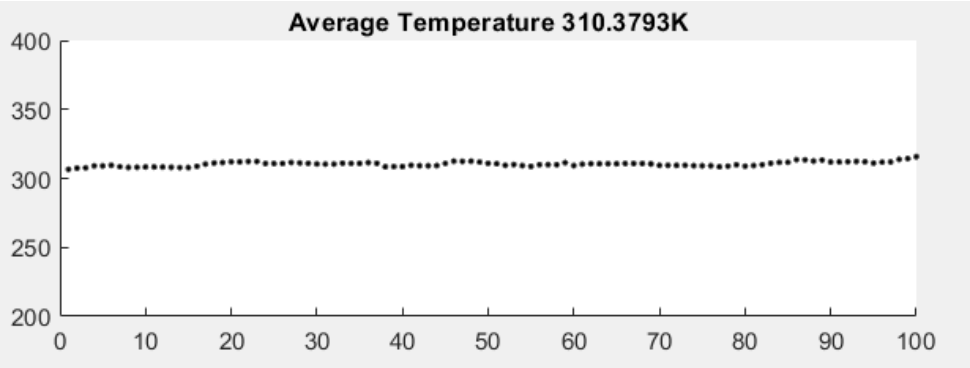

Figure 3: Average temperature in Semiconductor

From the above figure we can see that the average temperature varies very closely together. In part 1 we saw a fixed point of 300K while the plot above shows that the temperature never really is 300K but is very close and does not vary far off.

The Mean Free Path is followed throughout the simulation and included with the trajectory plot. The last plot of the simulation shows a Mean Free Path of 17.9 e^{-6}. Rearranging the same equation, we use for MFP, we can solve the time between collisions at the end of the simulation to cross reference the time we are assuming. In doing so we get a value of $1e^{-10}$. Although a little off from the given value of $0.2e^{-12}$, this can be caused from the scattering probability and normalized velocities.

The final plot of part 2 is a histogram of the final velocities. It is assumed at the final point in the electrons trajectory it has been affected by scattering. Therefor, we are expecting to see a normalized histogram, figure 4 and 5 show the final velocities in X and Y respectively.

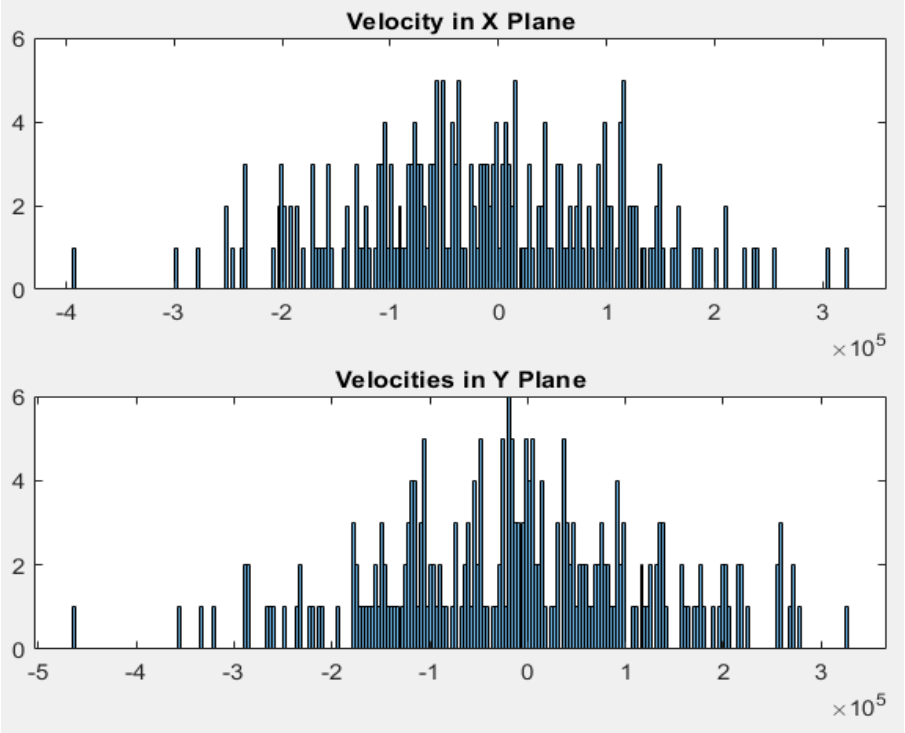

Figure 4: Histogram of Velocities

Above we can see that over time the velocities have normalized a few times with multiple peaks.

Part 3: Enhancements

The last part of the assignment required that further bounds be introduced into the area already used. The electrons will be required to treat these bounds the same as they treat the outer bounds and reflect off. We will also be adding a condition to treat the boundaries as either specular or diffusive and lastly, we will plot a density plot for the electrons final position as well as a temperature map using the same logic.

For adding the bounds, each electron was initially checked that their starting position was not inside of the boxes, the electrons initialized inside the boxes were randomly placed until no electrons were starting in the bounds. After this the simulation could enter the same loop as part 1 and 2 used. After the timestep however, the electrons are checked whether their path is leading toward a bound, if so they are reflected the appropriate direction. The code for checking the initial positions is shown below.

```
      OutBounds=((Pos(:,1)>L/4)&(Pos(:,1)<3*L/4))&((Pos(:,2)<W/4)|(Pos(:,2)>3
      *W/4));      %logical array for all values inside boxes.
      initialX=Pos(:,1);
      initialY=Pos(:,2);
     %the electrons initialized inside the boxes need to be randomly placed
     %again until no atoms initialize inside the box
       while (sum(OutBounds)>1)
           initialX(OutBounds)=initialX(OutBounds) .*randn();
%randomize the electrons initialized inside the box
           initialY(OutBounds)=initialY(OutBounds) .*randn();
```

```
            OutBounds=(initialX>L/4 & initialX<3*L/4)&(initialY<W/4
    |initialY>3*W/4);    %check once again for electrons initialized inside
    the box.
     End
```

the first line of code checks for the electrons that are inside the boxes and stores the positions in a logical array. The positions inside the boxes are scattered and checked again whether they are inside the boxes and this is continued until no electrons are starting inside the boxes.

```
Xless=((NextX <3*L/4 & NextX>L/4) &(NextY<W/3 | NextY>3*W/4)); %logical array
for values that are going into box
% below is where we check bounds of boxes
if sum(Xless) ~=0                                              %if there are
any values in the Xless array we have an electron on the boundary of the box
if (initialX(Xless)<3*L/4 & initialX(Xless)>L/4)            %check for
electron coming from the Y plane, reverse velocity in Y direction if so
Pos(Xless,4)=-Pos(Xless,4);
else                                                           %if its not
coming from Y plane that means its in x plane, reverse X velocity
Pos(Xless,3)=-Pos(Xless,3);
end
end
```

The positions to where the electrons are moving is needed to be checked as well. The electrons moving towards a bound are checked whether they are coming from a vertical or horizontal position and reflected through a change of velocity. The result of this is shown in figure 5 below.

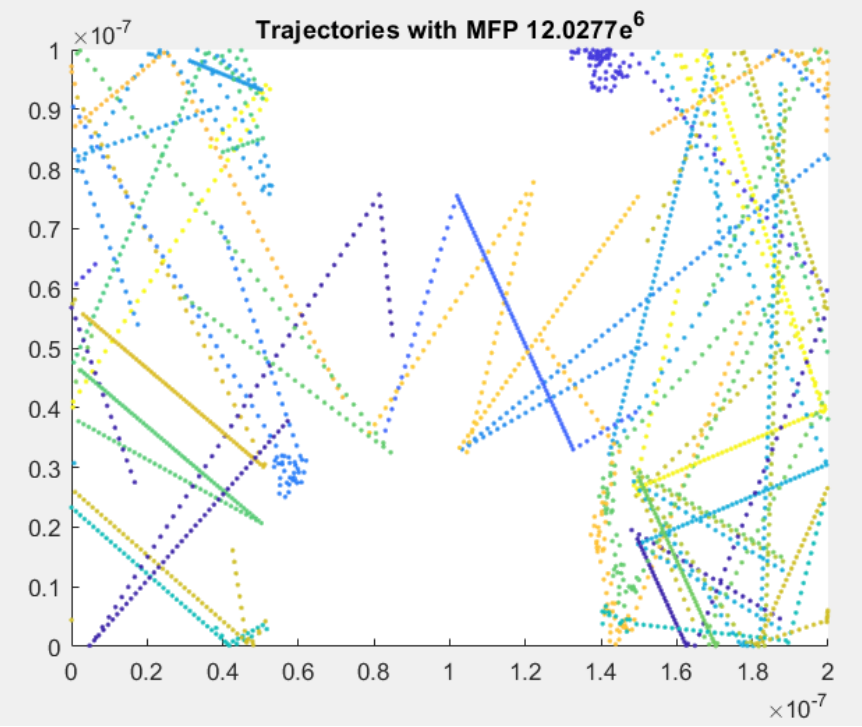

Figure 5: Trajectories with Enhanced Bounds

It is also required that the bounds change from specular or diffusive. To accomplish this there is an added requirement that a randomly generated number is rounded to the nearest integer to represent either specular or diffusive. Where 0 is specular and 1 is diffusive. Inside the large loop there is an added

IF statement to checks which bound reflection we are following. If we get specular our regular code is added inside however, when the other is being used the directional velocity must be randomized to simulate a diffusive boundary. The code for diffusive is shown below.

```
OverX=NextX > L;                 %check for values in next position that are
greater than bounds
NextX(OverX)=NextX(OverX)-L;          %subtract from the bound to reflect
Pos(OverX,3)=Vth*cos(randn()*2*pi);
Pos(OverX,4)=Vth*sin(randn()*2*pi);

UnderX=NextX < 0;                    %check for values less than bounds
NextX(UnderX)= NextX(UnderX) + L;    %reflect bound
Pos(UnderX,3)=Vth*cos(randn()*2*pi);
Pos(UnderX,4)=Vth*sin(randn()*2*pi);

OverY=NextY > W;                     %check for values greater then bounds
NextY(OverY)= 2*W -NextY(OverY) ;    %subtract from bounds to reflect
Pos(OverY,3)=Vth*cos(randn()*2*pi);
Pos(OverY,4)=Vth*sin(randn()*2*pi);

UnderY= NextY < 0;                   %check for under bounds
NextY(UnderY)=- NextY(UnderY);       %reflect
Pos(UnderY,3)=Vth*cos(randn()*2*pi);
Pos(UnderY,4)=Vth*sin(randn()*2*pi);
```

The last requirement in part 3 is to have a 3D representation of the Density of final electron position along with the temperature map. To complete this matrix is created to represent each grid point of the area with the cells representing a 10^{th} of the length and width. Vectors are created holding the values for the coordinates and each electron is then checked for where its final position is. Each position is logged into matrix to hold the number of electrons in that location. The final matrices are then plotted in 3D using surf, the code is shown below.

```
%for density, divide the L and W into a grid, step through grid points
    %to see how many electrons are in each grid point
    [X Y]=meshgrid(0:L/10:L,0:W/10:W);  %grid for stepping by 1/10 of width
and length
    %created an 11 x 11 vectors
    Dense=zeros(11,11);                 %will hold the density per nm
    Temp=zeros(11,11);                  %holds temperature per nm
    Tcount=0;                           %initilaize to 0
    Dcount=0;
for i=1:10
        XB1=X(1,i);             %holds the value of first bound grid
coordinate
        XB2=X(1,i+1);           %holds the value of the outside grid
coordinate
        for j =1:10
            YB1=Y(j,1);         %holds the value of the first Y bound
            YB2=Y(j+1,1);       %holds the value of the last Y bound

            %check each frame
            for k=1:PopE
                %Check to see if particle is within the bounds stored above
```

```
                if((Pos(k,1)>XB1 & Pos(k,1)<XB2) & Pos(k,2)<YB2 &
Pos(k,2)>YB1)
                    Tcount=Tcount+1;                                    %if it is
within the bound we update out temp counter
                    Dense(i,j)=Dense(i,j)+1;                            % we also
add to the density array count
                    Dcount=Dcount+sqrt(Pos(k,3)^2+Pos(k,4)^2);          % denisty
count.
                    if(Tcount >0)
                        Temp(i,j)=Mn*(Dcount^2)/(Tcount*K);             %the temp
is divided for each electron
                    end
                end
            end
            Dcount=0;
            Tcount=0;
        end
end
    %density map
figure(3)
    surf(X,Y,Dense)
    colorbar
    title 'Electron Density Map';
    zlabel 'Number of Electrons per Grid Point';
    ylabel 'Y Coordinate';
    xlabel 'X coordinate';
    %temperature map
    figure(4)
    surf(X,Y,Temp)
    colorbar
    title 'Temperature Density Map';
    zlabel 'Temperature per Grid Point';
    ylabel 'Y Coordinate';
    xlabel 'X coordinate';
```

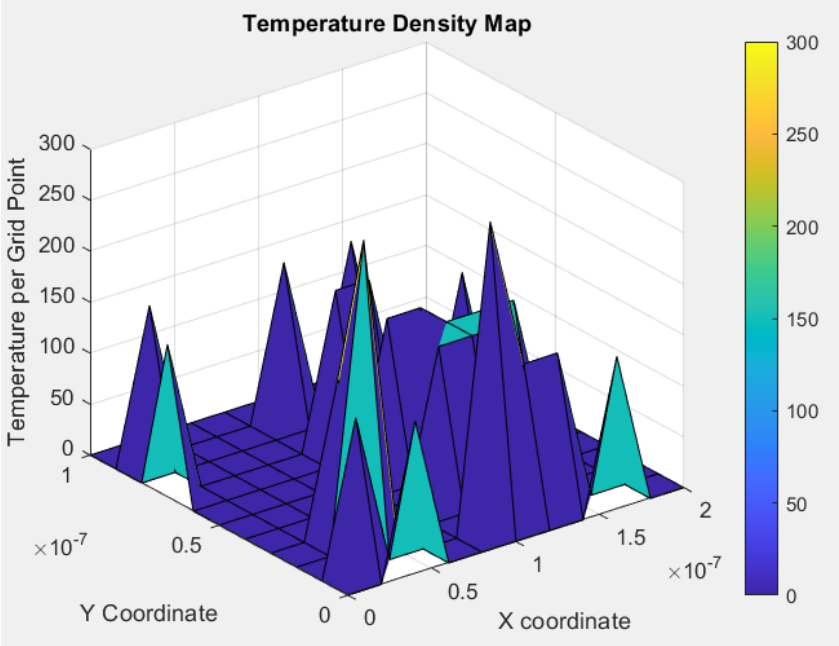

Figure 6: Temperature Density Map

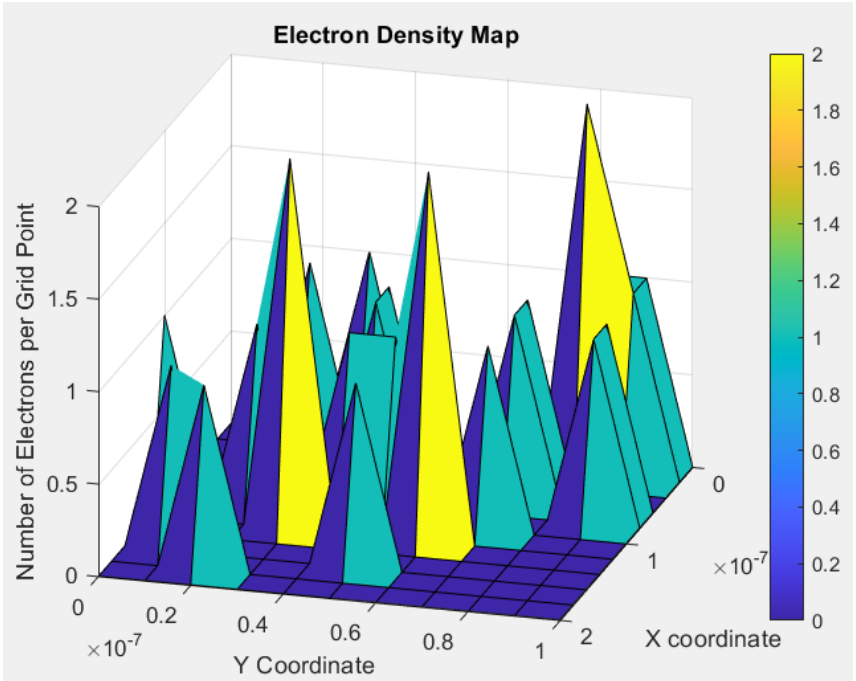

Figure 7: Electron Density Map

Conclusion:

In conclusion of this assignment we have successfully simulated the response of electrons in a semiconductor given different scenarios. The plots were correct representations of what we expect the behaviour to be. The final form of code is submitted on GitHub at https://github.com/NealeSkinner/Assignment1.git.