Ryan Lindsay 101038101 2/7/2021

1 Part 1: Electron Modelling

In part 1, a Monte-Carlo method was used to to create a simplistic simulation of electrons in a semiconductor. Electrons are first randomly assigned a position in a 200nm x 100nm boundary and then randomly assigned a fixed velocity but in a random direction. The next step is to create a periodic boundary in the x direction and a specular boundary in the y direction. In order to begin, a position vector was created that stores 4 components, the X and Y position and the X and Y velocity of each particle. The following block of code initializes the position of the electrons in the semiconductor

```
Ang = rand(Pop,1)*2*pi;  % Defines a random angle

Pos = [rand(Pop,1)*L rand(Pop,1)*W Vth*cos(Ang) Vth*sin(Ang)];  %Creates an Array of particles with random X & Y positions and velocities

initX = Pos(:,1); %The Initial X positions

initY = Pos(:,2); % The initial Y positions
```

The thermal velocity and MFP can be calculated using the following equations.

$$Vth = \sqrt{\frac{kT}{Mn}} = 1.322e^5$$

$$MFP = Tmn * Vth = 2.644e{-}^8$$

The next block of code creates the next position for the electron to travel to. It does this by multiplying the position by a given timestep. The second half of this block of code is the logic for checking the boundaries of the substrate. Logic was created such that if the Y position of any particle is greater than W, the Y velocity would be reversed, while preserving the X velocity. A similar idea was used for the left and right boundaries, except the X position of the electron was either added or subtracted by L in order to get it to the other side of the substrate. The entire block is included in a for loop in order to continuously determine the next position of the electrons in the substrate.

```
for i = 1 : lengthE      % Main Loop of the Function

    newX = initX + Pos(:,3)*Tstep;    % The next X position of the particle

    newY = initY + Pos(:,4)*Tstep;     % The next Y position of the particle


    % Checking for Top and Bottom bounds

    Yhigh = newY > W;
    newY(Yhigh) = 2*W - newY(Yhigh);
    Pos(Yhigh,4) = -Pos(Yhigh,4);

    Ylow = newY < 0;
    newY(Ylow) = -newY(Ylow);
    Pos(Ylow,4) = -Pos(Ylow,4);



    % Checking for Left and Right Bounds


    Xright = newX > L;
    newX(Xright) = newX(Xright) -L;


    Xleft = newX < 0;
    newX(Xleft) = newX(Xleft) + L;
```

A plot of the movement of electrons as well as a plot of the temperature can be seen in the figure below.

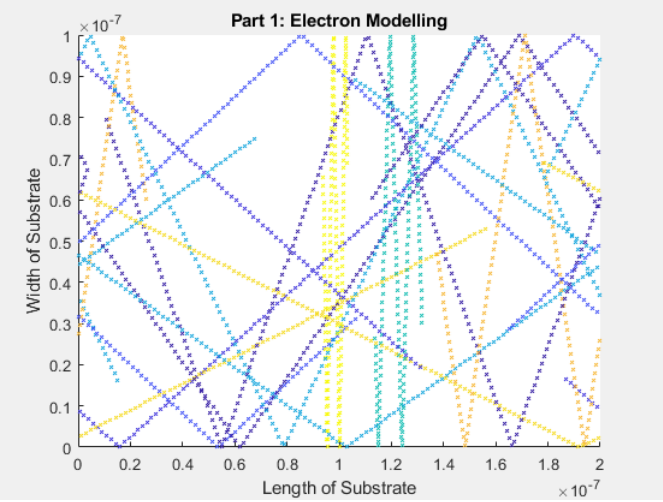

Figure 1: 2D Trajectories

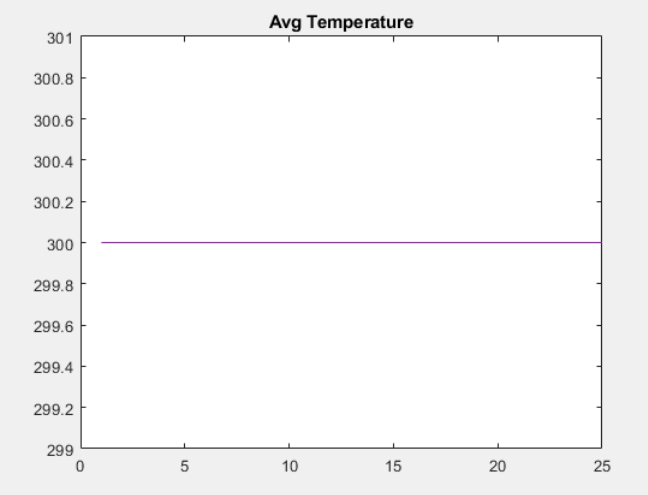

Figure 2: Average Temperature

2 Part 2: Collisions with Mean Free Path

Include in Report a) Histogram b) 2-D plot of particle trajectories c) Temperature plot d) MFP and mn

In Part 2, a random velocity was assigned to each particle with an average velocity of Vth. The scattering of each electron was modelled using an exponential scattering probability Pscat. When scattering occurs, the electron is re thermalized, and a new velocity is assigned to it. The base of the program remains similar to that of part 1, except each electron has a random chance of scattering in every loop. The block of code that enables this is shown below.

```
%%%%% NEW to Part 2 %%%%%%%%%

initT = T;

Pscat = 1- exp(-Tstep/Tmn);

Velo = sqrt(sum(Pos(:,3).^2)/Pop + sum(Pos(:,4).^2)/Pop);

probV = makedist('Normal', 'mu', 0, 'sigma', sqrt(K*T/Mn));

sumT = 0;


for i = 1 : lengthE      % Main Loop of the Function

    % Probability of scattering

    P = rand(Pop,1) < Pscat;

    Pos(P,3:4) = random(probV, [sum(P),2]);
```

The following figure shows the 2D particle trajectories with scattering. A scatter plot with dots was used to better show the different velocities of the particles. A larger gap in the distance between circles indicates a larger velocity. Also it can be seen that several of the particles have successfully scattered. The MFP was also calculated throughout the loop by taking the average velocity of the substrate each iteration.

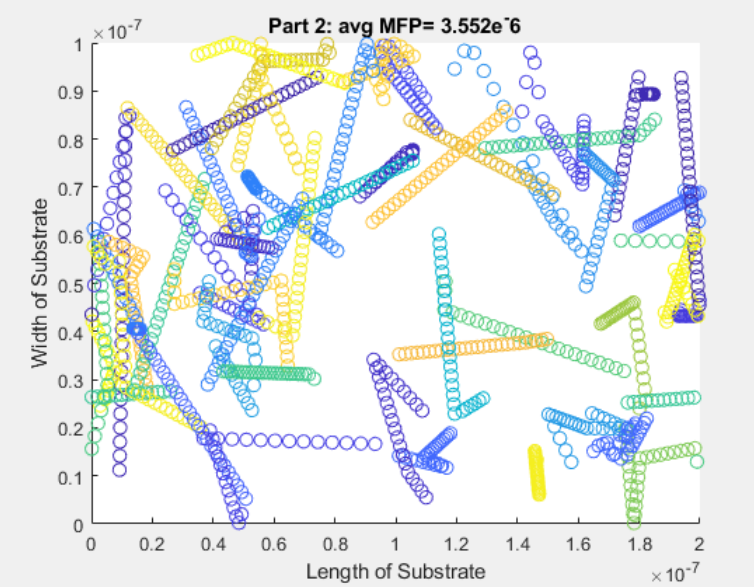

Figure 3: 2D Trajectories

The temperature of the system was measured by rearranging the thermal velocity equation in order to solve for temperature. The following snippet of code was used to record the temperature as the system changed in the loop.

```
% Finding velocity and Temperature
Velo = sqrt(sum(Pos(:,3).^2)/Pop + sum(Pos(:,4).^2)/Pop);

newT = T + ((Mn * (Velo.^2) )/K/Pop/2);
```

Using these equations, the temperature of the system was able to be plotted. This plot can be seen in the figure below. The average temperature seemed to fluctuate a small amount over 300K, but remained relatively consistent.

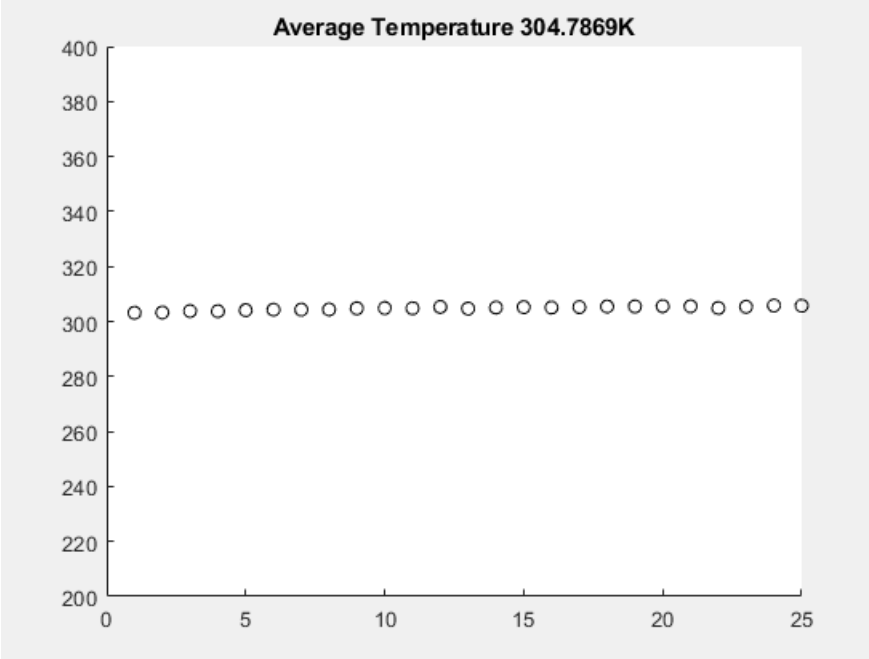

Figure 4: Average Temperature

The final portion of part 2 is a histogram of the velocities in both the X and Y direction. A normal distribution is expected. This distribution can be seen in the figure below. Note to achieve a reasonable normal distribution, the population of the substrate had to be increased.

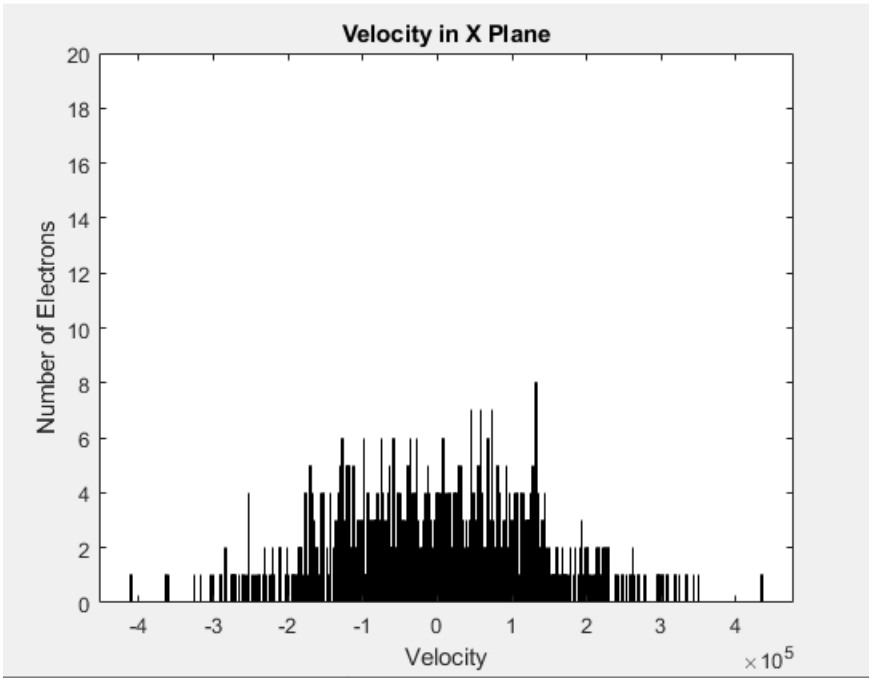

Figure 5: X Velocity

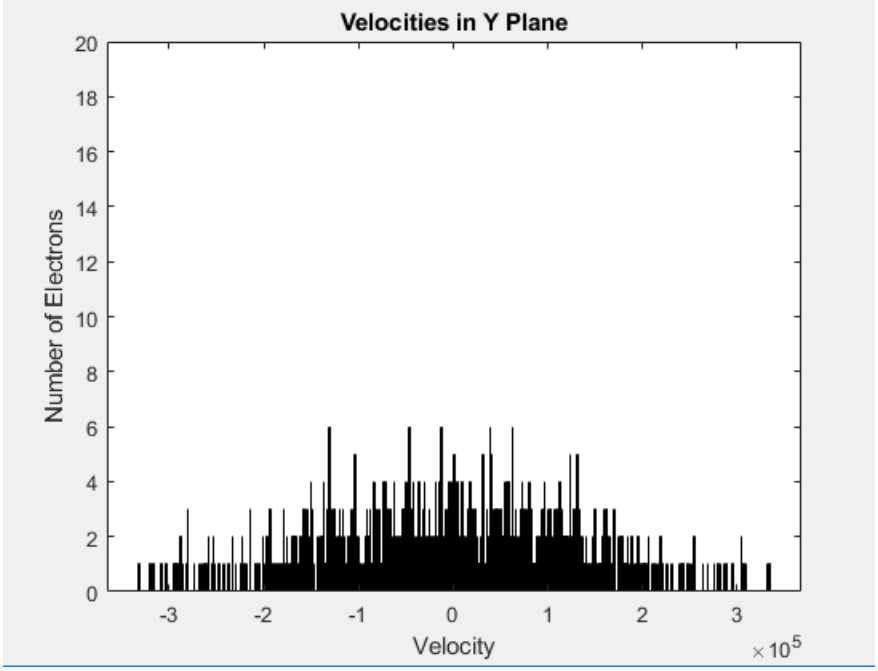

Figure 6: Y Velocity

3 Part 3: Enhancements

In Part 3, several alterations were made to the Part 2 code. Two boxes were added at the top and bottom of the substrate that reflect incoming electrons. In addition to this, there is now a probability that the electrons will be diffusive, meaning that will bounce off in a random direction. This functionality was also added to the top and bottom of the substrate, however the left and right sides remain the same as before. The first step is to establish the two boxes to act as boundaries. One issue is ensuring that each electron does not spawn within one of these boundaries. The following snippet of codes creates our boundaries, and ensures that each electron does not initially fall within it.

```
% Need to assign position outside of Boundaries

Bounds = ((Pos(:,1)>= L/3)&(Pos(:,1)<= 2*L/3)&(Pos(:,2)<= W/3)) | ((Pos(:,1)>= L/3)&(Pos(:,1)<= 2*L/3)& (Pos(:,2)>= 2*W/3));

while (sum(Bounds)>1)

          initX(Bounds)=initX(Bounds) .*rand();                          %randomize the electrons initialized inside the box

          initY(Bounds)=initY(Bounds) .*rand();


          Bounds = (initX>=L/4 & initX <=2*L/3 & initY <= W/3) | (initX>=L/4 & initX <=2*L/3 & initY >= 2*W/3); %check once again
      end
```

Now that the boxes have been created and the electrons have spawned successfully, we need to makes these boxes act as collision surfaces for the electrons to bounce off of. To begin a random generator was used to determine if the collision would be specular or diffusive. If the collision is specular, the following snippet of code would be used.

```
Yhigh = newY > W;
newY(Yhigh) = 2*W - newY(Yhigh);
Pos(Yhigh,4) = -Pos(Yhigh,4);

Ylow = newY < 0;
newY(Ylow) = -newY(Ylow);
Pos(Ylow,4) = -Pos(Ylow,4);



% Checking for Left and Right Bounds


Xright = newX > L;
newX(Xright) = newX(Xright) -L;


Xleft = newX < 0;
newX(Xleft) = newX(Xleft) + L;



% Checking for Box collisions


bCol = (newX>=L/3 & newX <=2*L/3 & newY <= W/3) | (newX>=L/3 & newX <=2*L/3 & newY >= 2*W/3); % Definition of the two boxes


if sum(bCol) ~= 0       % if bCol contains a value, we have an electron at a barrier

    if ((newX(bCol) >= L/3) & newX(bCol) <= 2*L/3);     % Checking X plane
        Pos(bCol,4) = -Pos(bCol,4);


    else
        Pos(bCol,3) = -Pos(bCol,3);    % Checking Y plane


    end
end
```

Essentially, a logical array is used to determine if an electron is at a boundary. If it is, either the X or Y velocity is reversed, depending on where the electron is coming from. For a diffusive collision, the same process is closely followed, except a random velocity is given instead of simply reversing it. The diffusive code can be seen below.

```
else

    % Checking for Top and Bottom bounds

Yhigh = newY > W;
newY(Yhigh) = 2*W - newY(Yhigh);
Pos(Yhigh,3)=Vth*cos(randn()*2*pi);
Pos(Yhigh,4)=Vth*sin(randn()*2*pi);



Ylow = newY < 0;
newY(Ylow) = -newY(Ylow);
Pos(Ylow,3)=Vth*cos(randn()*2*pi);
Pos(Ylow,4)=Vth*sin(randn()*2*pi);



% Checking For Box Values

    bCol = (newX>=L/3 & newX <=2*L/3 & newY <= W/3) | (newX>=L/3 & newX <=2*L/3 & newY >= 2*W/3); % Definition of the two boxes


if sum(bCol) ~= 0       % if bCol contains a value, we have an electron at a barrier

    if ((newX(bCol) >= L/3) & newX(bCol) <= 2*L/3);     % Checking X plane
        Pos(bCol,4) = -Pos(bCol,4);
        Pos(bCol,3)=Vth*cos(randn()*2*pi);
        Pos(bCol,4)=Vth*sin(randn()*2*pi);

    else
        Pos(bCol,3) = -Pos(bCol,3);    % Checking Y plane
        Pos(bCol,3)=Vth*cos(randn()*2*pi);
        Pos(bCol,4)=Vth*sin(randn()*2*pi);

    end
end
```

Now that the boundaries of the substrate have been established, the electrons can be simulated. There was found to be a small amount of leakage into the boxes, but the boundaries are mostly effective. The 2D plot can be seen in the figure below.

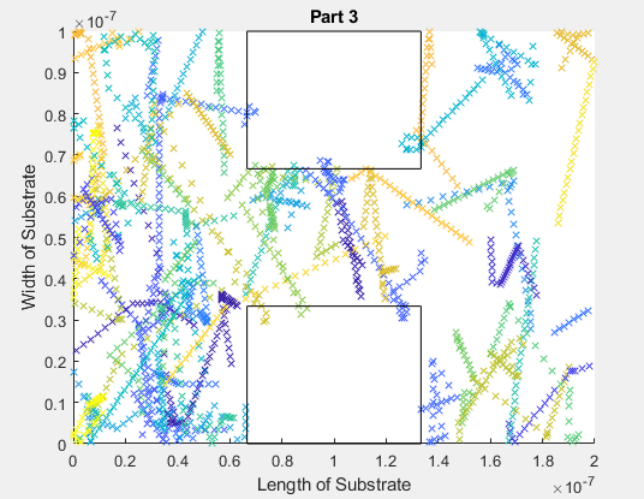

Figure 7: 2D Trajectories

The last portion of part 3 is the electron density map and the temperature map. To do this, a grid was created to split up the substrate in order to measure the temperature and count the number of electrons in each grid point. The following snippet of codes creates the grid and gathers the necessary information to plot both the temperature and the density of the substrate.

```
for i = 1:n


    XP1 = X(1,i);
    XP2 = X(1,i+1);

    for j = 1:n

        YP1 = Y(j,1);
        YP2 = Y(j+1, 1);


          %check each frame
            for k=1:Pop

                if((Pos(k,1)>XP1 & Pos(k,1)<XP2) & Pos(k,2)<YP2 & Pos(k,2)>YP1)

                    countT = countT+1;

                    Den(i,j) = Den(i,j)+1;

                    countD = countD + sqrt(Pos(k,3)^2+Pos(k,4)^2);


                    if(countT >0)
                        Temp(i,j)=Mn*(countD^2)/(countT)/K/2;
                    end
                end
            end

          %  countD=0;
           % countT=0;
    end
```

Using this logic, the temperature and density figures could be created.

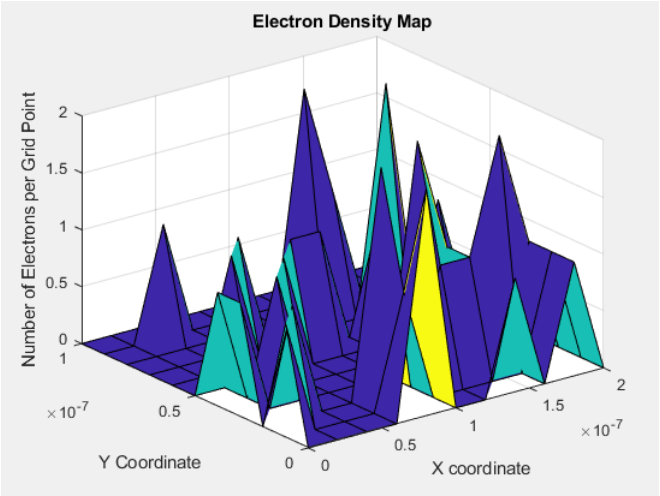

Figure 8: Electron Density Map

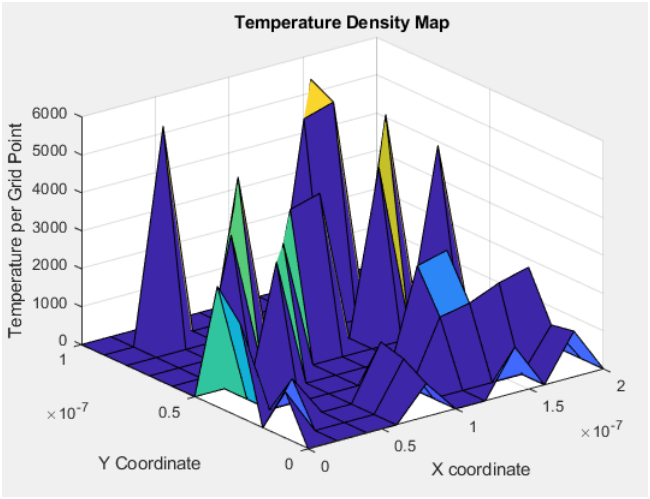

Figure 9: Temperature Density Map

In conclusion, the simulation was successful. Several different simulations were created and improved upon to increase the complexity of each the simulation.