Table of Contents

Ben Linton 100969769

Part 1 Electron Modelling

In this part I created a simple monte carlo electron model. The value of the thermal velocity is approximately 87,000 m/s and the formula is: $Vth = \frac{kB*T}{mn} = 8.7048 \times 10^4 m/s$.

Since the mean time between collisions is $Tmn = 0.2ps$ then the mean free path is:

$Dmn = Tmn * Vth = 1.741 \times 10^8 m$. Which is about 17 nm.

```
clear all;
close all;

global Vth dt j
```

Basic variables

```
m0 =9.11E-31; % electron mass
mn=0.6*m0; % effective electron mass
kB=1.3806E-23; % Boltzmann Constant
T=300; % system temperature
Vth=(kB*T/mn)^0.5; % Thermal Velocity
Tmn=0.2E-12; % Time between collisions
Dmn=0.2E-12*Vth;% Mean Free Path
```

Simulation Specific Variables

```
NumP = 1000; % Number of particles
MaxIt =200; % Maximum Iterations
ylimit=100E-9; %Vertical limit
xlimit=200E-9; %Horizontal limit
dt= ylimit/(Vth*100); % simulation time step
NumPP=10; % Number of particls to plot
```

The colors available for lines to be plotted in.

```
color=hsv(NumPP);
```

Temperature array for storing the temperature over time.

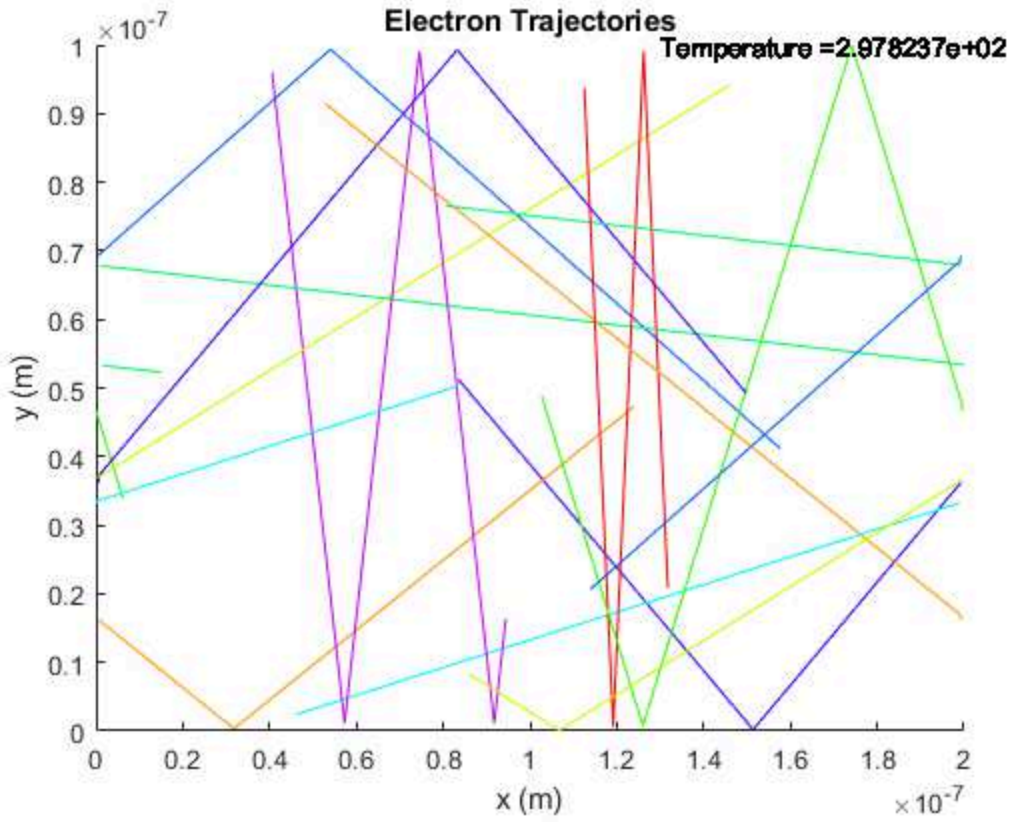
```
temp=zeros(1,MaxIt);
```

This loop creates electrons in an array titled 'electrons' and gives them an initial x,y,vx,and vy. It will create as many electrons as specificied by 'NumP'. It calls the function Celec which is the function that actually assigns initial conditions to the electron array.

```
for j=1:NumP
      electrons(j,:)=Celec();
      POS1(j,:)=POS(electrons);
      POS2(j,:)=POS(electrons);
end
```

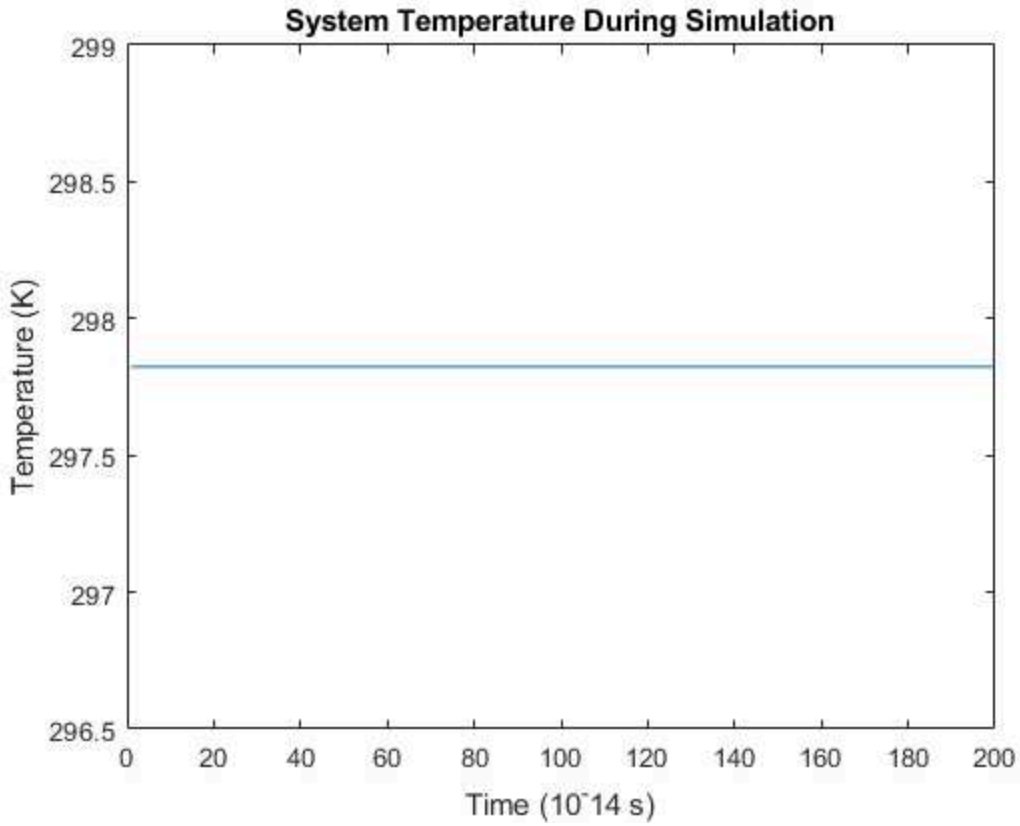
In part 1 the electron trajectory conditions are that only the top and bottom are relfective and that the sides are continuous. The electrons are moved by the move function and the system temperature is calculated and stored at each time interval. The move function uses logical indexing to properly enforce the continuous and reflective boundary conditions and update the electrons new positions in x and y according to their velocities vx and vy.

```
figure(1); % electron trajectory figure
xlim([0 200E-9]); % x axis limit
ylim([0 100E-9]); % y axis limit
hold on

for j=1:MaxIt
    electrons = move(electrons); % moves the electrons
    temp(j) = (sum(electrons(:,3).^2) + sum(electrons(:,4).^2))*mn/
(kB*2*NumP);
    for i=1:NumP
          if abs(POS1(i,1) - electrons(i,1))>100E-9;
            POS1(i,1)=electrons(i,1);
%In the case of cts right and left, do not plot across the screen.
          end
          if(i<NumPP) % Plots only the first 10 electrons
             plot([POS1(i,1) electrons(i,1)],[POS1(i,2)
 electrons(i,2)],'Color',color(i,:)) ;
%Plots a line between the old and new electron positions.

             %pause(0.00001)
          end
    end
    POS1(:,1)=electrons(:,1); % stores the electron X position.
    POS1(:,2)=electrons(:,2);% stores the electron Y position.
    text(130E-9,100E-9,sprintf('Temperature =%d',temp(j))) %
 continuously displays the temperature

end

title('Electron Trajectories') % assign a title
xlabel('x (m)') % assign x axis label to electron trajectory plot
ylabel('y (m)') % assign y axis label to electron trajectory plot
```



This code plots the system temperature. Which we can see is constant over the simulation.

```
tx = [1 MaxIt];
figure(2)
plot(tx,temp(tx));
title('System Temperature During Simulation')
xlabel('Time (10^-14 s)')
ylabel('Temperature (K)')
```



Part 2 Collisions with Mean Free Path (MFP)

```
clear all;
close all;

global Vth dt j Tmn eu Pscat NumP MFPX XSUM YSUM MFPY MFPL MFP MaxIt
 JSUM JXY
```

Constants and key variables

```
m0 =9.11E-31;
mn=0.6*m0;
kB=1.3806E-23;
T=300;
Vth=(kB*T/mn)^0.5;
Tmn=0.2E-12;
Dmn=0.2E-12*Vth;
eu=2.71828;
temp=zeros(1,MaxIt);
```

Simulation Specific Variables

```
NumP = 1000;
MaxIt =200;
ylimit=100E-9;
xlimit=200E-9;
```

```
dt= ylimit/(Vth*100);
NumPP =10;
color=hsv(NumPP);
```

The scattering proability $Pscattering = 1 - (eu^{-\frac{dt}{Tmn}})$

```
Pscat = zeros(NumP,1);
Pscat(:,1)= 1 - (eu^(-1*dt/Tmn));
%
```

Mean Free Path and Time matrices

```
XSUM =zeros(NumP,1); % Counter for x distances
MFPX=zeros(NumP,MaxIt); % Keeps track of x positions of scattered
 particles
YSUM =zeros(NumP,1); % Counter for y distances
MFPY=zeros(NumP,MaxIt);% Keeps track of y positions of scattered
 particles
MFPL=zeros(NumP,MaxIt); % Calculates the total distance
 L=sqrt(x^2+y^2)
MFP=zeros(NumP,1); % Stores all the Mean free paths in one matrix
JSUM=zeros(NumP,1); % Counter for scattered particles mean free time
JXY=zeros(NumP,MaxIt); % Stores all the times particles scatter at and
 how long they have been 'free' for
```

In part 2 the electrons must be given an initial velocity according to the boltzmann distribution. To do this Celec was modified to Celec2 which properly assigns velocities according to the boltzmann distribution.

The equation used to assign velocities is: $Vn = randn() \times \frac{Vth}{\sqrt{2}}$

By using the above equation we will pick velocities from a Maxwell Boltzmann distribution for the correct temeperature.

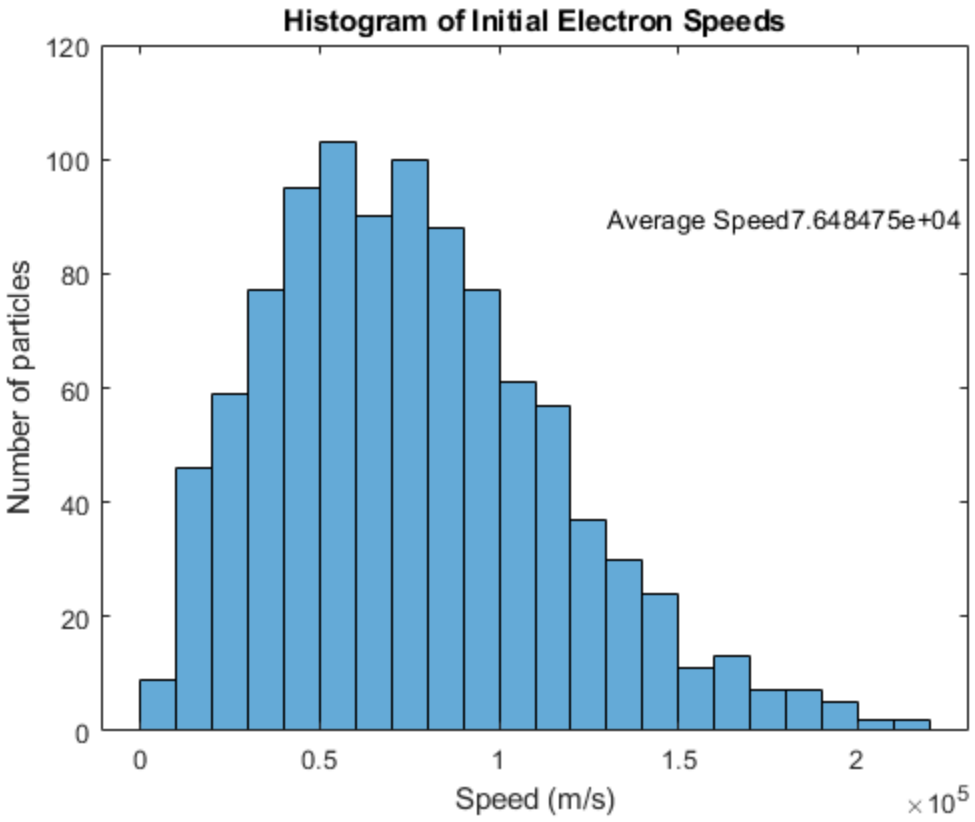
```
for j=1:NumP % creates electrons
      electrons(j,:)=Celec2();
      POS1(j,:)=POS(electrons);% Tracks initial position
      POS2(j,:)=POS(electrons);%Tracks initial position
end
```

I populate a vector with all the initial velocities of the electrons so that we can create a histogram of all the intial electron velocities. I also find the average so that we can display it to make sures we are within the correct range.

```
MB = sqrt(electrons(:,3).^2 + electrons(:,4).^2);
MBAv = sum(MB)/length(MB);
```

This code generates the required histogram of initial electron velocities.

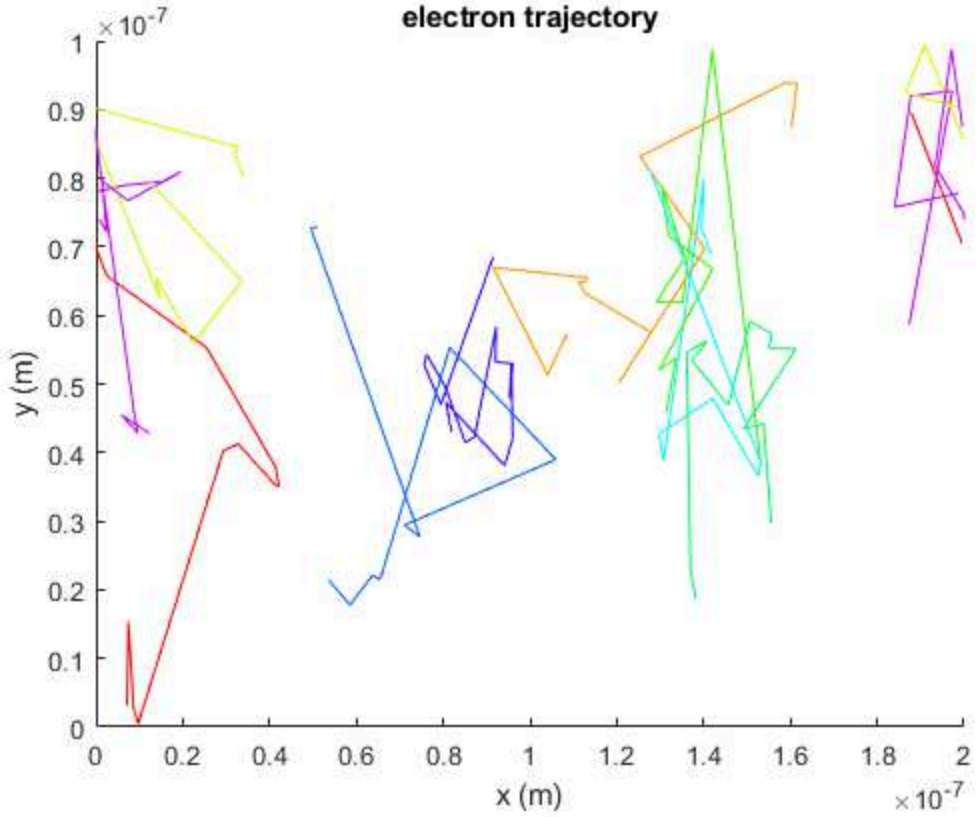
```
figure(3)
histogram(MB);
title('Histogram of Initial Electron Speeds');
xlabel('Speed (m/s)');
ylabel('Number of particles');
text(1.3E5,90,sprintf('Average Speed%d',MBAv)) % displays the average
 of the histogram
pause(1)
```



In part 2 the same boundary conditions apply however now the particles have the ability to scatter and instaneously change their direction and velocities. To account for this the function 'move2' was adapted from the function in part 1 'move'. Asides from scattering the move function now also tracks the scattering locations and times so that we can calculate the mean free path and times of the electrons. Again we set up the electron trajectory plotting.

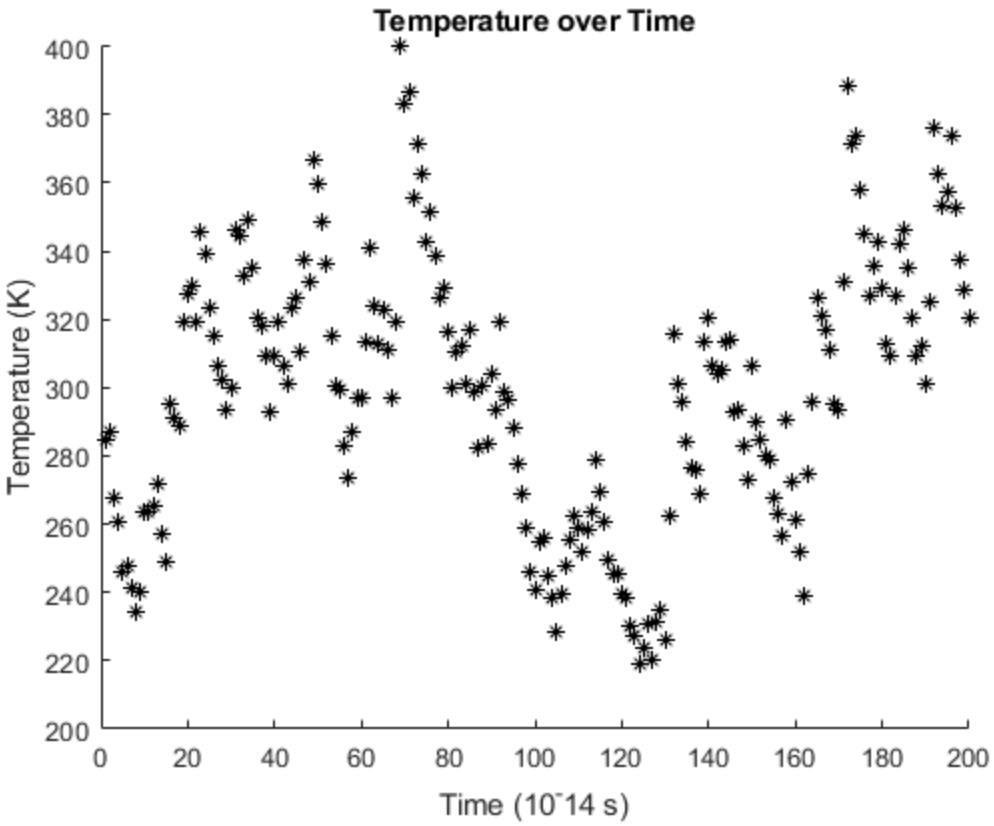
```
figure(4)
xlim([0 200E-9]);
ylim([0 100E-9]);
hold on

for j=1:MaxIt
    electrons = move2(electrons);
    temp(j) = (sum(electrons(:,3).^2) + sum(electrons(:,4).^2))*mn/
(kB*NumP);
    for i=1:NumP
          if abs(POS1(i,1) - electrons(i,1))>100E-9;
            POS1(i,1)=electrons(i,1);
          end
          if(i<NumPP)
             plot([POS1(i,1) electrons(i,1)],[POS1(i,2)
 electrons(i,2)],'Color',color(i,:))
             %pause(0.0001)
          end
    end
    POS1(:,1)=electrons(:,1);
```

```
    POS1(:,2)=electrons(:,2);

end

title('electron trajectory')
xlabel('x (m)')
ylabel('y (m)')
```



For the temperature plot I would have liked to plot a line but but it seems matlab is only letting me plot stars or dots.Unsure why.

```
figure(5)
hold on
for i=1:200
plot(i,temp(i),'*k');
end
title('Temperature over Time')
xlabel('Time (10^-14 s)')
ylabel('Temperature (K)')
text(130E-9,100E-9,sprintf('Temperature =%d',temp(j)))
```

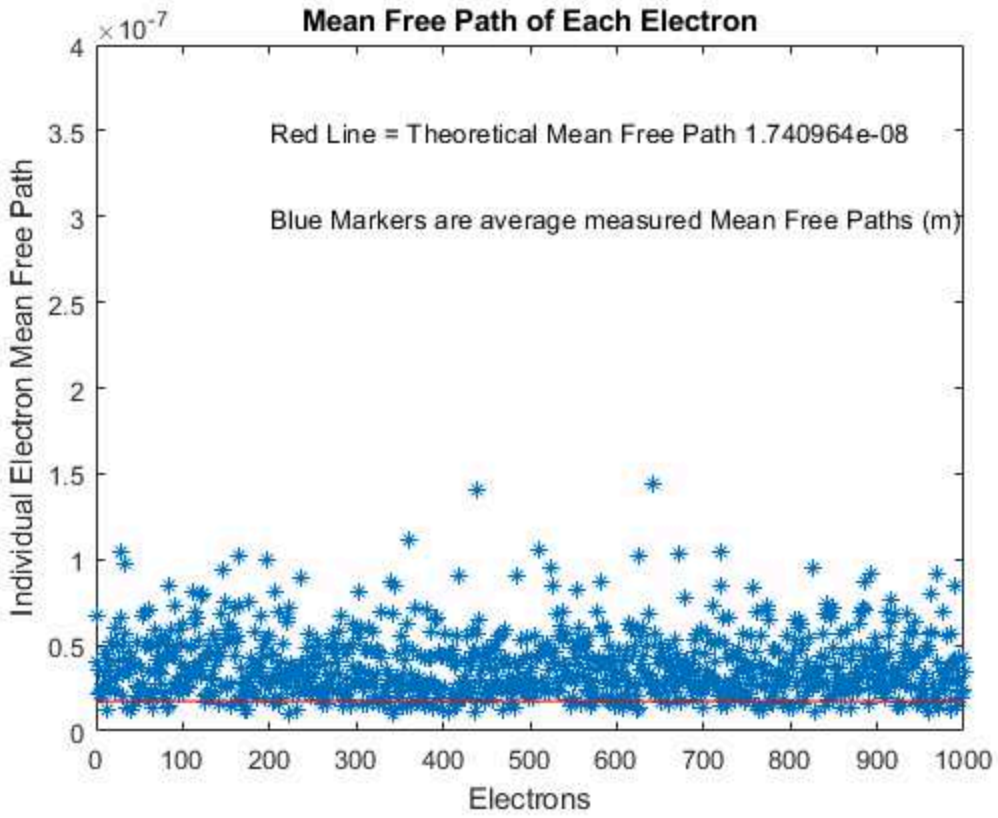


Mean free path and time calculations stored in the vectors MFP and MFT

```
MFP =mfp(MFPX,MFPY);
MFT=mft(JXY);
```

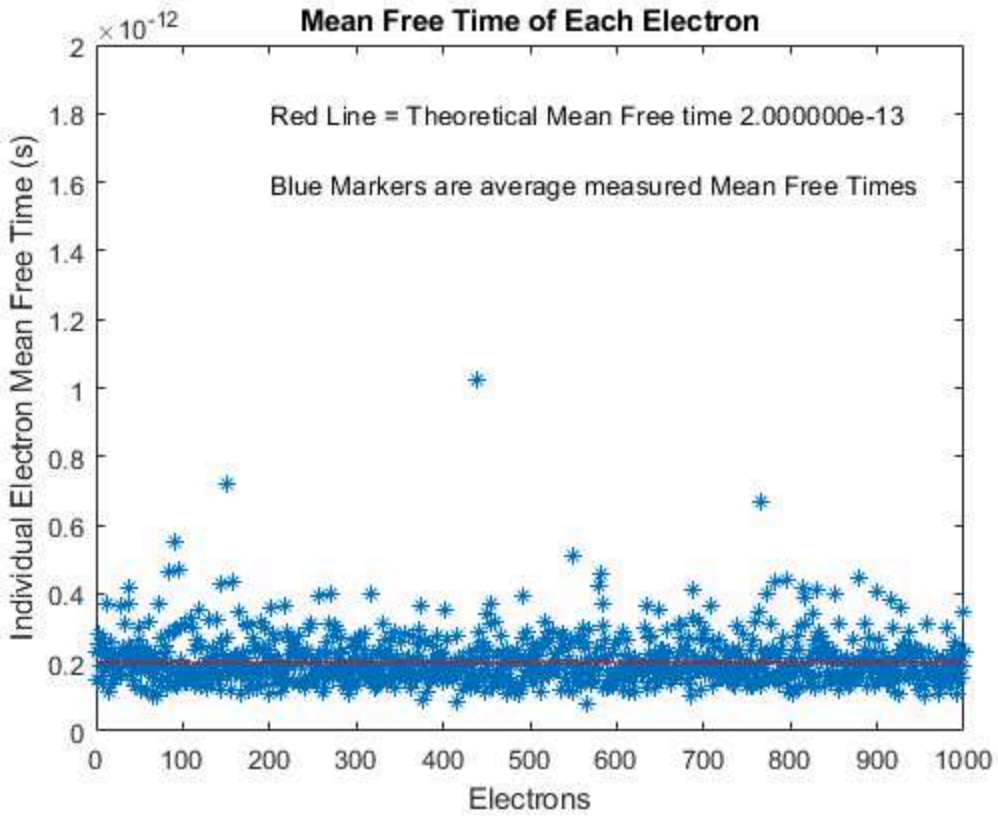
The plot illustrates the measured MFP's in blue stars and the theoretical MFP in a red line as calculated in part 1

```
figure(6)
plot(MFP,'*');
hold on
line([0 1000],[Dmn,Dmn],'Color','red','LineStyle','-');
title('Mean Free Path of Each Electron')
xlabel ('Electrons')
ylabel('Individual Electron Mean Free Path')
xlim([0 NumP]);
ylim([0 400E-9]);
text(200,350E-9,sprintf('Red Line = Theoretical Mean Free Path
 %d',Dmn)) % displays the temperature of the system on the plot
text(200,300E-9,sprintf('Blue Markers are average measured Mean Free
 Paths (m)'))
```



The plot illustrates the measured MFT's in blue stars and the theoretical MFT as a red line as given in part 1

```
figure(7)
plot(MFT,'*');
hold on
line([0 1000],[Tmn,Tmn],'Color','red','LineStyle','-');
title('Mean Free Time of Each Electron')
xlabel ('Electrons')
ylabel('Individual Electron Mean Free Time (s)')
xlim([0 NumP]);
ylim([0 20E-13]);
text(200,1.8E-12,sprintf('Red Line = Theoretical Mean Free time
 %d',Tmn)) % displays the temperature of the system on the plot
text(200,1.6E-12,sprintf('Blue Markers are average measured Mean Free
 Times'))
```



Part 3 Enhancements

```
clear;
close all

global Vth dt j Tmn eu Pscat NumP  MaxIt boxes specular diffusive ax1
 ax
```

Constants and basic variables

```
m0 =9.11E-31;
mn=0.6*m0;
kB=1.3806E-23;
T=300;
Tmn=0.2E-12;
Dmn=0.2E-12*Vth;
eu=2.71828;
temp=zeros(1,MaxIt);
```

Simulation specific variables

```
NumP = 1000;
MaxIt =200;
ylimit=100E-9;
xlimit=200E-9;
Vth=(kB*T/mn)^0.5;
```

```
dt= ylimit/(Vth*100);
NumPP=10; % # of electrons to plot
color=hsv(NumPP);

ax =zeros(NumP,1);
ax1=zeros(NumP,1);
```

Specular or diffusive options

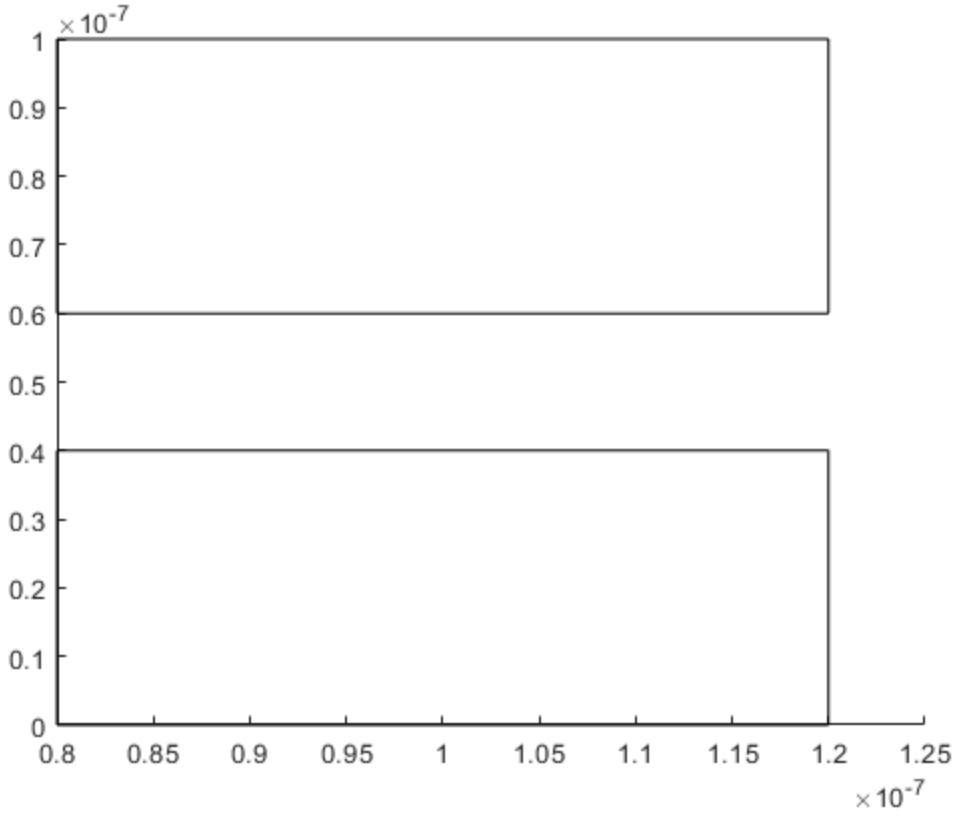
```
specular=0;
diffusive=1;
```

Scattering probability

```
Pscat = zeros(NumP,1);
Pscat(:,1)= 1 - (eu^(-1*dt/Tmn));
```

The boxes are created and drawn on the figure. The array the boxes are stored in will be used in the 'isbox' function later, that checks if electrons are in a box or not.

```
figure(8)
hold on
boxes = 1e-9.*[80 120 0 40; 80 120 60 100];

for j=1:size(boxes,1)
           plot([boxes(j, 1) boxes(j, 1) boxes(j, 2) boxes(j, 2)
 boxes(j, 1)],...
                [boxes(j, 3) boxes(j, 4) boxes(j, 4) boxes(j, 3)
 boxes(j, 3)], 'k-');
end
```



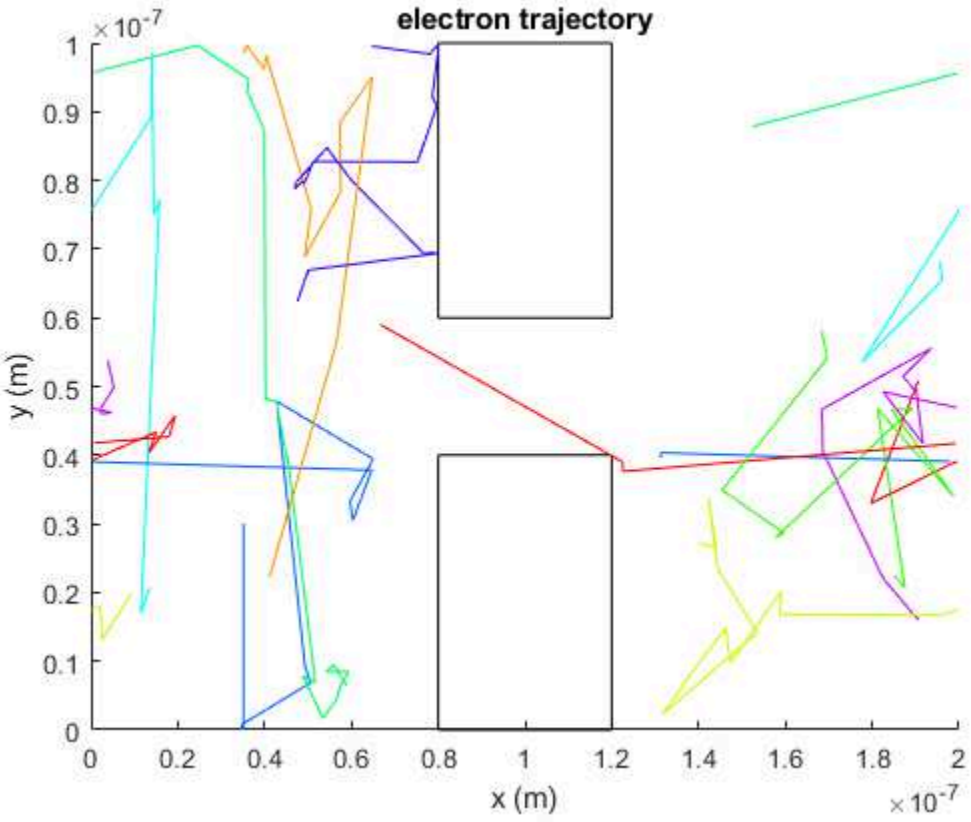
To properly create electrons the function Celec3 was created which is like Celec2 and Celec. However the function uses another function 'isbox' to check if a electron is in a box and if so moves the electron to another position until it is not in a box anymore.

```
for j=1:NumP % creates electrons
      electrons(j,:)=Celec3();
      POS1(j,:)=POS(electrons);% Tracks initial position
      POS2(j,:)=POS(electrons);%Tracks initial position
end
```

In Part 3 we must also deal with the addition of the boxes and specular and diffusive scattering. To do this I created a function 'logixbox' which returns a logical indexing vector for each electron saying if it will enter a box or not on the next iteration. I used this logical index to apply proper reflection conditions (specular or diffusive for the electrons. In addition I created logical indexes for diffusive and specular so that reflections off the top and bottom are a choice of specular or diffusive depending on the initial choice made by the user. To make all this happen the function 'move3' was created as well as the 'logixbox' function.

```
figure(8)
xlim([0 200E-9]);
ylim([0 100E-9]);
hold on

for j=1:MaxIt
    electrons = move3Final(electrons);
    %temp(j) = (sum(electrons(:,3).^2) + sum(electrons(:,4).^2))*mn/
(kB*NumP);
    for i=1:NumP
```

```
            if abs(POS1(i,1) - electrons(i,1))>100E-9;
              POS1(i,1)=electrons(i,1);
            end
            if(i<NumPP)
               plot([POS1(i,1) electrons(i,1)],[POS1(i,2)
 electrons(i,2)],'Color',color(i,:))
              % pause(0.0001)
            end
    end
    POS1(:,1)=electrons(:,1);
    POS1(:,2)=electrons(:,2);

end

figure(8)
title('electron trajectory')
xlabel('x (m)')
ylabel('y (m)')
```



To properly create particle density and temperature maps, I created two vectors to store the temperature and density information.

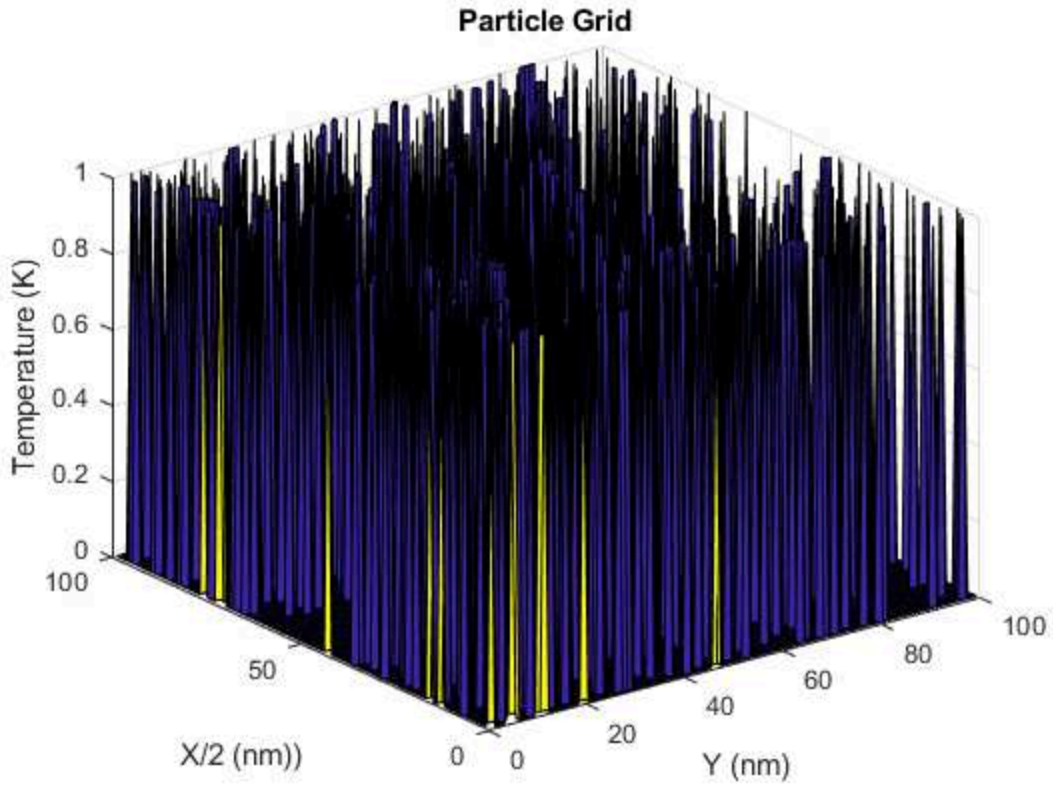
```
particle_grid = zeros(100,100);
temp_grid = zeros(100,100);
```

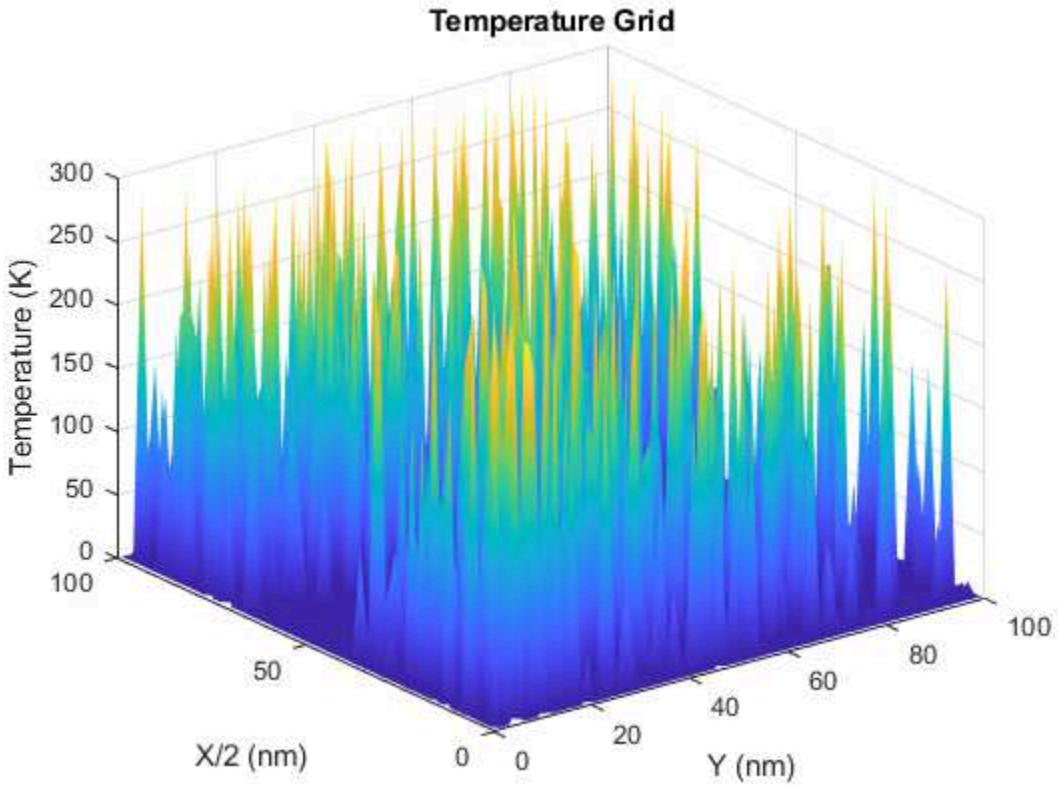
The electrons are counted and tallied into regions in the matrix 'particle grid'. The temperature of each region is calcualted and stored in the matrix 'temp_grid.

```
xlimit = 200e-9;
ylimit = 100e-9;

for x=1:100
    for y=1:100
        for u = 1:NumP
            if((electrons(u,1) <= (xlimit*(x/100))) && (electrons(u,1)
 > (xlimit*((x-1)/100))) && (electrons(u,2) <= (ylimit*(y/100))) &&
 (electrons(u,2) > (ylimit*((y-1)/100))))
                particle_grid(x,y) =+ 1;
                temp_grid(x,y) =+ (electrons(u,4)^2)*mn/(2*kB);
            end
        end
    end
end

for u=1:100
    temp_grid(u,:) = mod(temp_grid(u,:),1000);
    temp_grid(u,:) = mod(temp_grid(u,:),10000);
    temp_grid(u,:) = mod(temp_grid(u,:),100);
    temp_grid(u,:)=temp_grid(u,:)*3;
end
```

The plots for the electron density and temperature are created. However they are given in 3-D view so I manually added some 2-d plot images.

```
figure(10)
surf(particle_grid)
xlim([0 100]);
ylim([0 100]);
xlabel('Y (nm)')
ylabel('X/2 (nm))')
zlabel('Temperature (K)')
title('Particle Grid')

figure(11)
xlim([0 100E-9]);
ylim([0 100E-9]);
surf(temp_grid)
shading interp
xlabel('Y (nm)')
ylabel('X/2 (nm)')
zlabel('Temperature (K)')
title('Temperature Grid')
```


Particle Grid
Temperature (K)
1
0.8
0.6
0.4
0.2
0
100
50
0
X/2 (nm))
0
20
40
60
80
100
Y (nm)


Temperature Grid
Temperature (K)
300
250
200
150
100
50
0
100
50
0
X/2 (nm)
0
20
40
60
80
100
Y (nm)

Published with MATLAB® R2018b