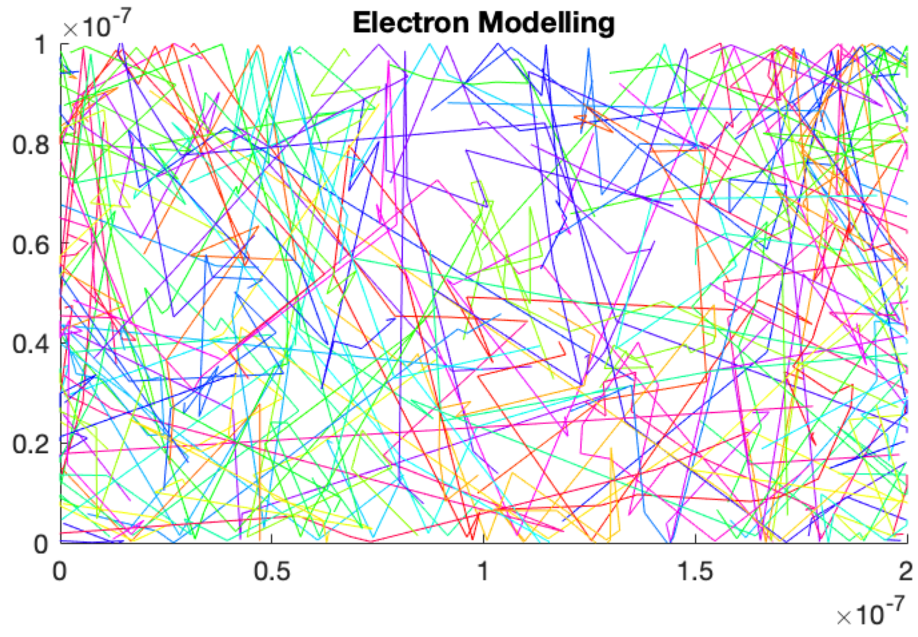
Assignment 1: Part 2

The second part of the assignment included several realistic modifications to the code. The first update was on the velocity distribution assigned to the electrons. The first part involved assigning a uniform velocity distribution based on the calculated thermal velocity. The second part assigns the velocities using the Maxwell-Boltzman distribution, which is a gaussian distribution in each direction with a standard deviation of the thermal velocity previously calculated. Random scattering was also introduced to the system. Based on a scattering probability given in the assignment, particles would be assigned a random direction. When the particle scatters, the velocity was re-thermalized and new velocities were assigned once again using the Maxwell-Boltzman distribution. The particle trajectory plot can be found below.

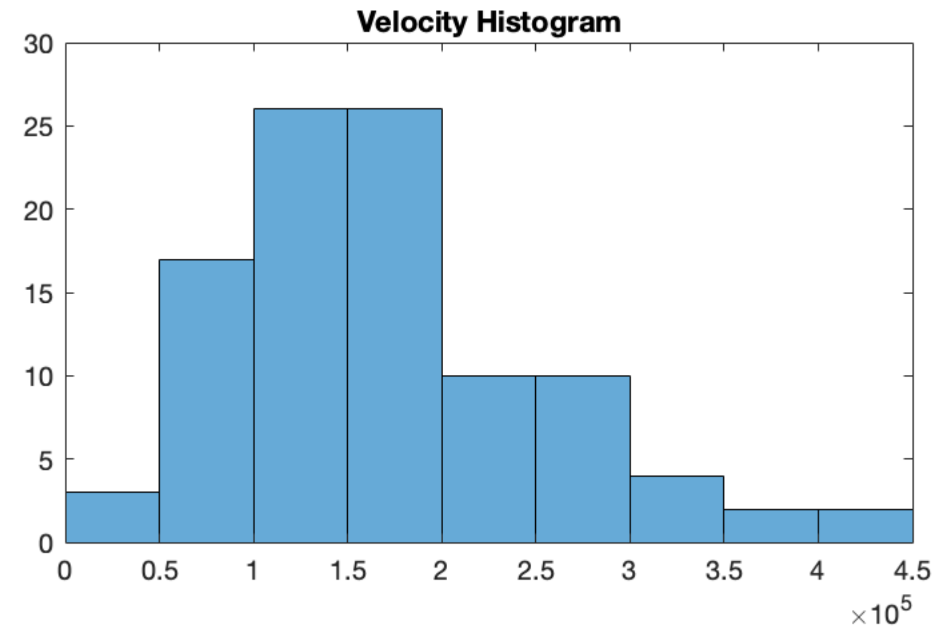
```
close all

%Part 2, Add collisions with MFP
% constants
T = 300; % temperature in Kelvin
m0 = 9.11E-31;
mn = 0.26*m0;
kb = 1.38E-23;
vth = sqrt((2*kb*T)/mn);


%region limits
xlim = 200E-9;
ylim = 100E-9;

num_particles = 100;
%initialize all of the particles
% 1 - x
% 2 - y
% 3 - direction (angle)
% 4 - vth
particle_vector = zeros(num_particles, 4);

%past position matrix for creating a line plot
past_position = zeros(num_particles, 2);

%Setting up the timing for the plot
%time step should be smaller than 1/100 of region size
time_step = 1E-14;
total_time = time_step * 100;
number_steps = total_time/time_step;
colour = hsv(num_particles);
time = 0;
l = 0;

%Temperature
temp = zeros(number_steps, 1);

%Used for plotting temperature
time_array = zeros(number_steps, 1);
```

```
%for histogram
velocity_array = zeros((number_steps*num_particles), 1);

%scatter function
Pscat = 1 - exp((-time_step)/(0.2E-12));

%variables to calculate time between collisions
%collision_time = zeros (number_steps, 1);
temp_avg = 0;

%loop to assign an initial position and fixed velocity
for i=1:1:num_particles
    %loop assigning the matrix
    for j=1:1:4
        if (j==1)
            %random x position
            particle_vector(i, j) = xlim*(rand(1));
        elseif (j==2)
            %random y position
            particle_vector(i, j) = ylim*(rand(1));
        elseif (j==3)
            %random direction
            particle_vector(i, j) = 2*(pi)*(rand(1));
        else
            %PART 2: Velocity is now random
            %Use Maxwell Boltzman Distrubution

            Vx = (vth*randn())/1.41;
            Vy = (vth*randn())/1.41;
            particle_vector(i, j) = sqrt(Vx^2 + Vy^2);
        end
    end
end
collision_counter = 0;
%loop that updates particles position with respect to each time step
for m=0:time_step:total_time
   temp_avg = 0;
   l = l+1;

    %for loop to update each particle
    for n=1:1:num_particles

        %handle all of the boundary conditions
        %x boudary conditions - particle jumps to other side
        if (particle_vector(n, 1)>=xlim)
            particle_vector(n, 1) = 0;
            past_position(n, 1) = 0;
        elseif (particle_vector(n, 1) <= 0)
            particle_vector(n, 1) = xlim;
            past_position(n, 1) = xlim;
        end

        %y boundary conditions - particle reflects at the same angle
        if
(((particle_vector(n,2)+particle_vector(n,4)*sin(particle_vector(n,3))*time_step)
ylim) || ((particle_vector(n,
2)+particle_vector(n,4)*sin(particle_vector(n,3))*time_step)<= 0))
            particle_vector(n, 3) =  pi - particle_vector(n, 3);
            particle_vector(n, 4) = - particle_vector(n, 4);
        end

        if (Pscat > rand())
           %then particle scatters (new direction and velocity)
           %assign random direction
           particle_vector(n, 3) = rand()*2*pi;
           %assign new velocity THIS IS USES GAUSSIAN - USE MAXWELL
BOLTZMAN
            Vx = (vth*randn())/1.41;
            Vy = (vth*randn())/1.41;
            particle_vector(i, j) = sqrt(Vx^2 + Vy^2);
           %store the time for collisions
           collision_counter = collision_counter + 1;

        end

        %create the plot
        if (m~=0)
            figure(1)
            plot([past_position(n, 1),particle_vector(n, 1)],
[past_position(n, 2), particle_vector(n, 2)], 'color', colour(n, :));
            axis([0 xlim 0 ylim]);
        end

        %velocity_array((l*n), 1) = particle_vector(n, 4);
        temp_avg = temp_avg + (((particle_vector(n,4))^2)*mn)/(2*kb);

    end


    %set past position equal to current position
    past_position (:, 1) = particle_vector(:, 1);
    past_position (:, 2) = particle_vector(:, 2);

    %update the current x position taking into account the velocity
    particle_vector(:,1) = particle_vector(:,1) +
particle_vector(:,4).*cos(particle_vector(:, 3)).*time_step;
    %update the current y position taking into account the velocity
    particle_vector(:,2) = particle_vector(:,2) +
particle_vector(:,4).*sin(particle_vector(:, 3)).*time_step;

    title 'Electron Modelling';
    hold on;
    pause(0.001);

    time = time + time_step;
    %Calculate SemiConductor Temperatures
    %temp(l, 1) = ((particle_vector(1,4)^2)*mn)/(2*kb);
    temp(l,1) = (temp_avg/(num_particles+1));
    time_array(l, 1) = time;

end
```



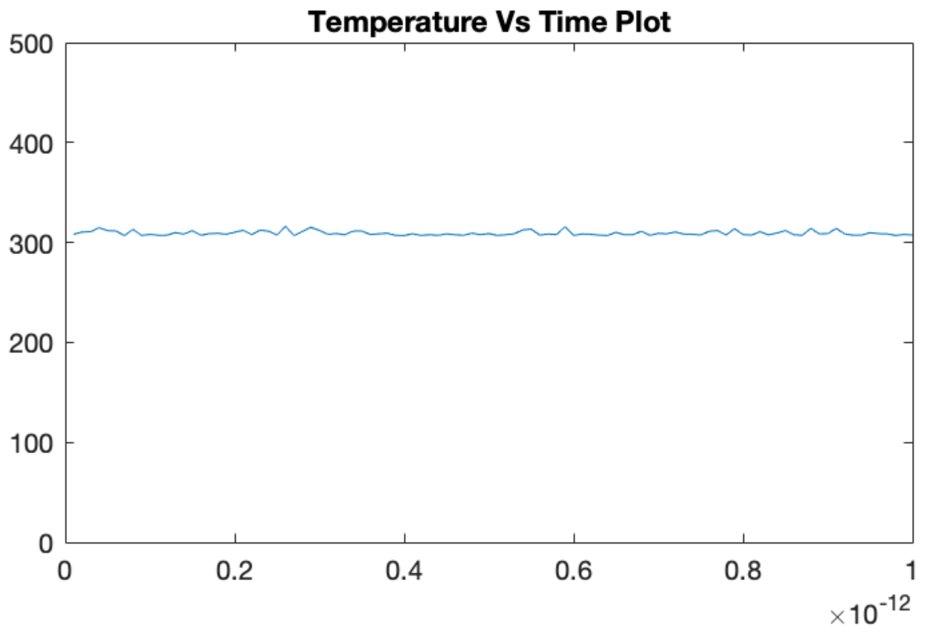
A histogram of the particle velocities was created. The histogram demonstrated the Maxwell-Boltzman distribution. The distribution would be more noticeable for more particles and a greater amount of time.

```
velocity_array = zeros(num_particles, 1);

for i=1:num_particles
   velocity_array(i,1) = abs(particle_vector(i, 4));
end

figure (2)
histogram(velocity_array);
title 'Velocity Histogram'
```



As well, a temperature vs time plot was created to display how the temperature deviated over time. It was noted that there was now deviations in the temperature due to the scattering and no longer uniform velocity distribution.

```
figure(3)
plot(time_array, temp);
axis([0 total_time 0 500])
title 'Temperature Vs Time Plot';
```



The collision mean and Mean Free Path (MFP) were also found. The results can be seen found below. The mean time between collisions was found to be approzimately 2×10^-15s and the MFP was found to be 0.56ns.

```
collision_mean = total_time/collision_counter;
MFP = collision_mean*vth;
%mean
```

Published with MATLAB® R2017a