Assignemt 3: Part 1

The first part of the assignment consisted of modifying the Monte Carlo simulation code created in Assignment 1, part 2. This code consisted of a Monte Carlo simulation monitoring the position and movement of particles with a given mass ina defined region of space. In the third assignment, an electric field is introduced into the region of space.

A voltage of 0.1V is applied across the semi-conductor, purely in the x direction. The electric field acting upon the electrons is therefore defined as the voltage divded by the area that the voltage would be acting upon. In this case, it was the width of the region of space.

$$EField = Voltage/(X - limit) = 5 \times 10^5 V/m$$

The force acting on each particle can be found by taking the product of the electric field and the charge of a single particle, q.

$$Force = EField * q = 8 \times 10^-14N$$

The acceleration on each particle can be found be dividing the force acting up each particle by the mass of each particle.

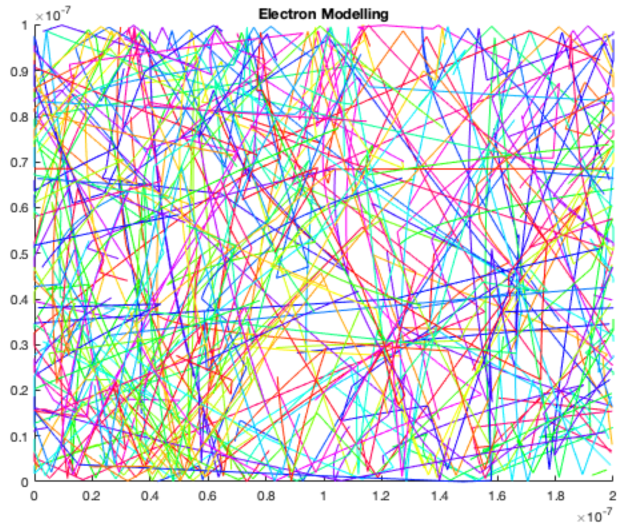
$$Acceleration = Force/m_n = 3.382 \times 10^17m/s^2$$

```
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;

voltage = 0.1;
efield = voltage/xlim;
force = efield*1.602E-19;
a = force/mn;
```

The Monte Carlo simulation was then recreated, taking into account how the electric field would act upon each particle. The velocity of each particle, specifically in the x-direction, takes into account the acceleration caused by the added electric field. The Results can be found in the plot below.

```
num_particles = 100;
%initialize all of the particles
% 1 - x
% 2 - y
% 3 - direction (angle)
% 4 - vth
% 5 - vx
% 6 - vy

particle_vector = zeros(num_particles, 6);

%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 = randn();
            Vy = randn();
            V_TH = vth*sqrt(Vx^2 + Vy^2);
            particle_vector(i, j) = V_TH;
            particle_vector(i, 5) = particle_vector(i,
 4).*cos(particle_vector(i, 3));
```

```
            particle_vector(i, 6) = particle_vector(i,
 4).*sin(particle_vector(i, 3));
            %particle_vector(i, j) = sqrt(Vx^2 + Vy^2);
            %particle_vector(i, j) = randn()*vth;
        end
    end
end
counter = 0;
drift_v = 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;
   counter = counter + 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
%
        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

        if (particle_vector(n, 1)<0)
            particle_vector(n, 1) = 0;
        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 (((particle_vector(n,2) + particle_vector(n,6)*time_step)>=
 ylim) || ((particle_vector(n, 2) + particle_vector(n,6)*time_step)<=
 0))
            particle_vector(n, 6) = - particle_vector(n, 6);
        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 = randn();
            Vy = randn();
            V_TH = vth*sqrt(Vx^2 + Vy^2);
            particle_vector(n, 4) = V_TH;
            particle_vector(n, 5) = particle_vector(n,
4).*cos(particle_vector(n, 3));
            particle_vector(n, 6) = particle_vector(n,
4).*sin(particle_vector(n, 3));

           %particle_vector(n, 4) = rand()*vth;
           %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

        drift_v = drift_v + particle_vector(n,5);
        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);

    %account for the electric field
    particle_vector(:,5) = particle_vector(:,5) + a*time_step;
    particle_vector(:,1) = particle_vector(:,1) + particle_vector(:,
5).*time_step;
    particle_vector(:,2) = particle_vector(:,2) + particle_vector(:,
6).*time_step;


    %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(0.01);

    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;

    drift(counter,1) = drift_v/num_particles;
    drift_i(counter,1) = q*1E15*drift(counter,1);

end
```

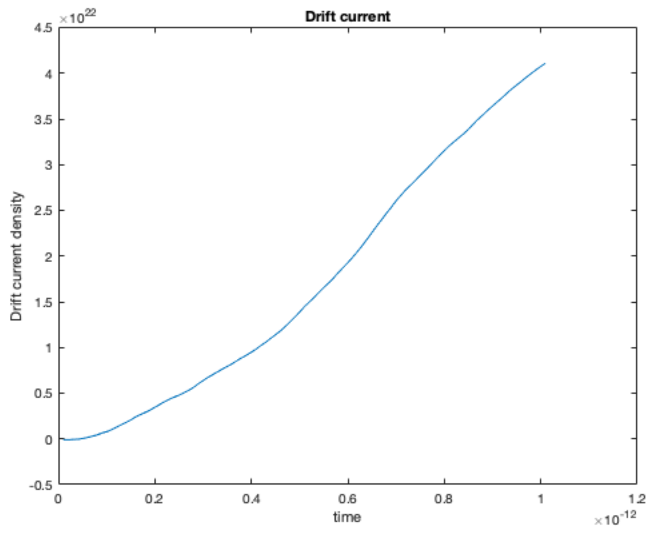


The relationship between drift current density and average carrier velocity was then determined. An assumed electron concentration of $(10^15)cm^(-2)$$ was used for the calculation. Each particle reflects the behaviour of a particle contraining the assumed electron contrentration.

$J_n = -enV_d$

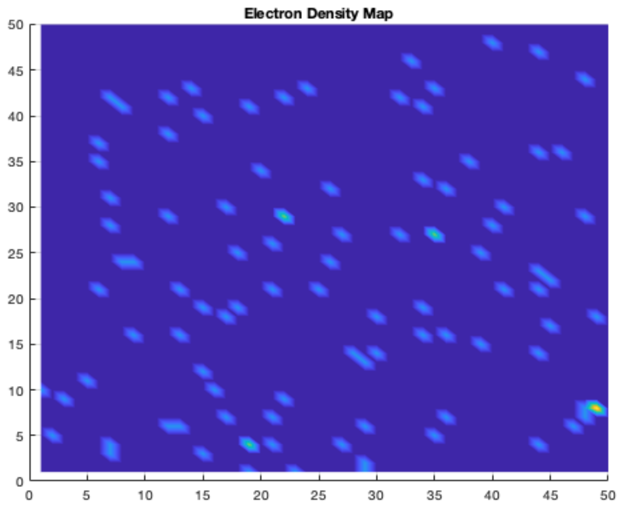
The drift current was calculated and plotted. The results can be seen in the figure below. It was noted that the drift current velocity increased with time. This is attributed to the acceleration which will increase the dift velocity with time.

```
figure(2)
plot(time_array,drift_i)
title('Drift current')
xlabel('time')
ylabel('Drift current density')
```



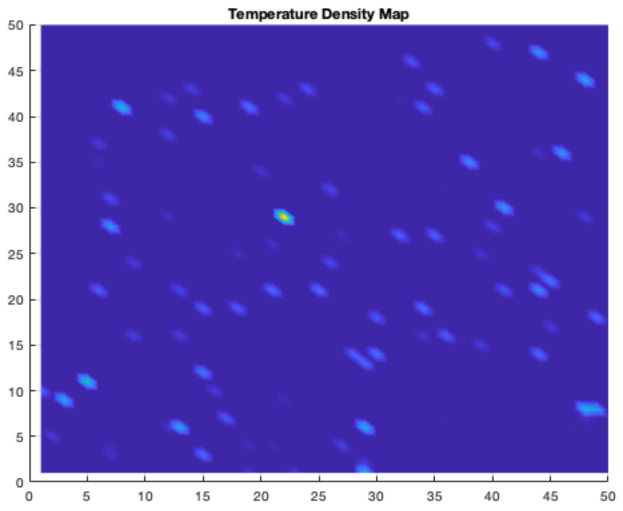
Finally, temperature and density maps were generated at the end of the simulation for a representation on where the particles were located and their temperatures at this time. The electron density map can be seen in the figure below.

```
elec_density = zeros(50,50);
temperature_density = zeros(50,50);
sizeparticle = size(particle_vector);
for Xc=1:50
    for Yc = 1:50
        for count = 1:sizeparticle
           if((particle_vector(count,
 1)<=((Xc/50)*xlim))&&(particle_vector(count, 1)>(((Xc -
 1)/50)*xlim)))
                if((particle_vector(count,
 2)<=((Yc/50)*ylim))&&(particle_vector(count, 2)>(((Yc -
 1)/50)*ylim)))
                    elec_density(Xc,Yc) = elec_density(Xc,Yc) + 1;
                    temperature_density(Xc,Yc)
 = ((particle_vector(count,4)^2)*mn)/(2*kb) +
 temperature_density(Xc,Yc);
                end
           end
        end
    end
end


figure(3)
[X, Y] = meshgrid(1:1:50,1:1:50);
%scatter(particle_vector(:,1),particle_vector(:,2), 'r', 'filled');
surf(elec_density)
shading interp
title('Electron Density Map');
view(0,90);
%hold on
```



The temperature density for the region was calcuted in the same loop that found the electron density map for the region. The temperature density map can be found in the figure below.

```
figure(4)
[A, B] = meshgrid(1:1:50,1:1:50);
surf(temperature_density)
shading interp
title('Temperature Density Map');
view(0,90);
```



Published with MATLAB® R2017a