Assignment 1: Part 3

The final stage of the assignment involved adding in two square regions that bottle neck the movement of the particles. Handling the regions involved ensuring no particles were spawned within the regions and defining each edge of the boxes as a border. The particles would bounce of the regions, similar to how they were to bounce off the top of the region. The simulation result can be found below.

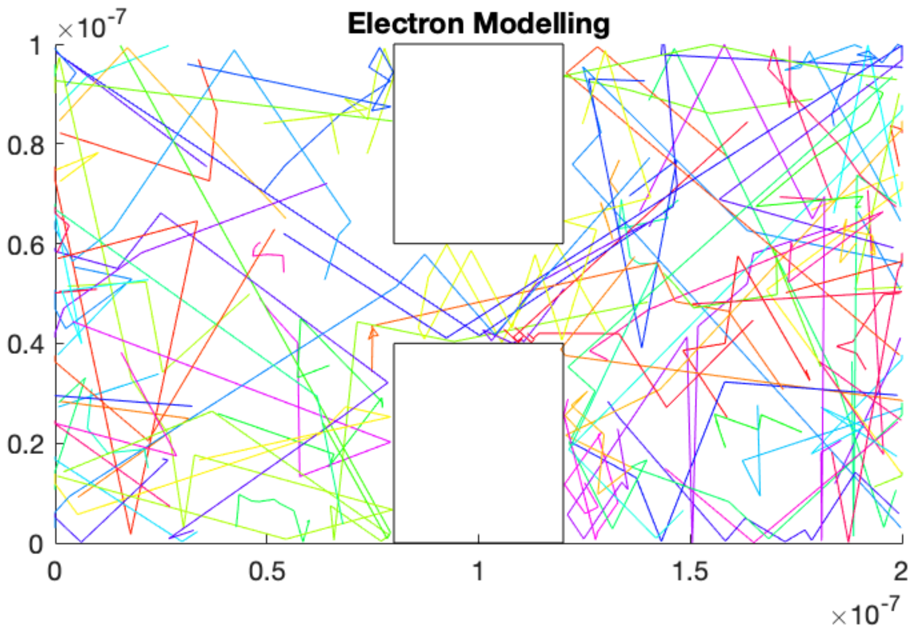
```
close all
T = 300;
m0 = 9.11E-31;
mn = 0.26*m0;
kb = 1.38E-23;
vth = sqrt((2*kb*T)/mn);

xlim = 200E-9;
ylim = 100E-9;

%box limits
x_low_lim = 8E-8;
x_high_lim = 12E-8;
y_low_lim = 4E-8;
y_high_lim = 6E-8;

num_particles = 50;

particle_vector = zeros(num_particles, 4);

%past position matrix
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;


temp = zeros(number_steps, 1);
time_array = zeros(number_steps, 1);
velocity_array = zeros((number_steps*num_particles), 1);

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

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));
        end

        %make sure no particle spawns in box regions
        while ((particle_vector(i, 1) >= x_low_lim) &&
 (particle_vector(i, 1) <= x_high_lim) && (particle_vector(i,
 2) >= y_high_lim))||((particle_vector(i, 1) >= x_low_lim) &&
 (particle_vector(i, 1) <= x_high_lim) && (particle_vector(i, 2) <=
 y_low_lim) && particle_vector(i, 2)~= 0)
            particle_vector(i, 1) = xlim*(rand(1));
            particle_vector(i, 2) = ylim*(rand(1));
        end

        if (j==3)
            %random direction
            particle_vector(i, j) = 2*(pi)*(rand(1));
        elseif(j==4)
            %PART 2 & 3: 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

%loop that updates particles position with respect to each time step
for m=0:time_step:total_time
   l = l+1; % This was pretty hacky - need a pointer for the array
 element in temp
    temp_avg = 0;
    %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

        %box cases


%         %x low case
         if(((particle_vector(n,2) >= y_high_lim)||
(particle_vector(n,2) <= y_low_lim)) && particle_vector(n, 1) < 85E-9)
             if
 ((particle_vector(n,1)+particle_vector(n,4)*cos(particle_vector(n,3))*time_step)
 >= x_low_lim)
                particle_vector(n, 3) = - particle_vector(n, 3);
                particle_vector(n, 4) = - particle_vector(n, 4);
             end
         end
%
%         %x high case
         if(((particle_vector(n,2) >= y_high_lim)||
( particle_vector(n,2)<= y_low_lim)) && particle_vector(n, 1)> 118E-9)
             if
 ((particle_vector(n,1)+particle_vector(n,4)*cos(particle_vector(n,3))*time_step)
 <= x_high_lim)
                particle_vector(n, 3) = - particle_vector(n, 3);
                particle_vector(n, 4) = - particle_vector(n, 4);
            end
         end

        %y high case
        if
 (((particle_vector(n,2)+particle_vector(n,4)*sin(particle_vector(n,3))*time_step)
 >= y_high_lim) && (particle_vector(n,1) >= x_low_lim &&
 particle_vector(n,1) <= x_high_lim && particle_vector(n, 2)<(61E-9)))
            particle_vector(n, 3) =  pi - particle_vector(n, 3);
            particle_vector(n, 4) = - particle_vector(n, 4);
        end

        %y low case
        if
 (((particle_vector(n,2)+particle_vector(n,4)*sin(particle_vector(n,3))*time_step)
 <= y_low_lim) && (particle_vector(n,1) >= x_low_lim &&
 particle_vector(n,1) <= x_high_lim && particle_vector(n, 2)
 >(39E-9)))
            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);

        end

        %create the plot
        if (m~=0)
            figure(1)
            %plot([particle_vector(n, 1), particle_vector(n, 2)],
 'color', colour(n, :));
            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]);
            rectangle('Position',[80E-9,60E-9,40E-9,40E-9]);
            rectangle('Position',[80E-9,0,40E-9,40E-9]);
        end
        temp_avg = temp_avg + (((particle_vector(n,4))^2)*mn)/(2*kb);
        %velocity_array((l*n), 1) = particle_vector(n, 4);

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

end

sizeparticle = size(particle_vector);
elec_density = zeros(50,50);
temperature_density = zeros(50,50);
%create density plots with regions
for Xc=1:50
    for Yc = 1:50
        for count = 1:sizeparticle
```

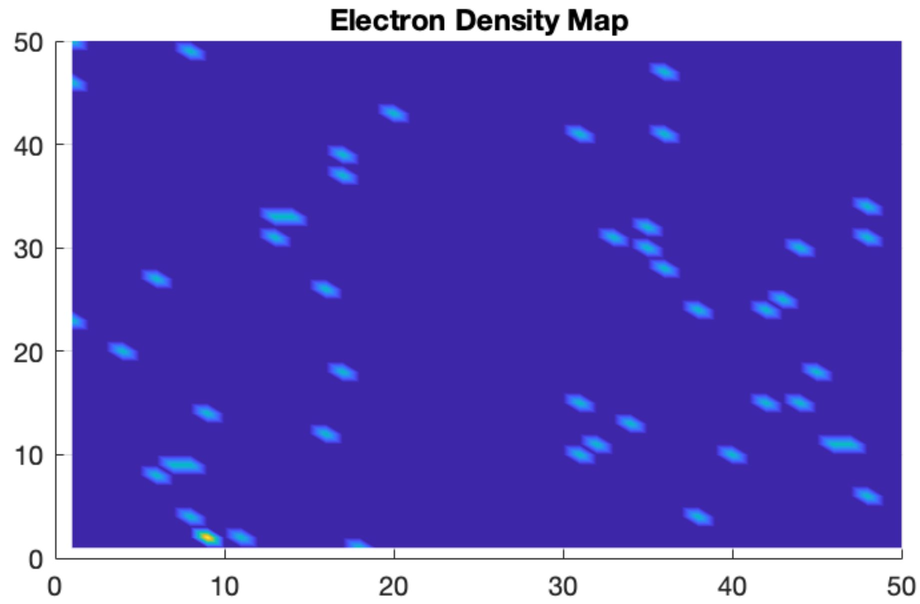
```
            if((particle_vector(count,
1)<=((Xc/50)*xlim))&&(particle_vector(count, 1)>(((Xc -
1)/50)*xlim)))
                if((particle_vector(count,
2)<=((Yc/50)*ylim))&&(particle_vector(count, 2)>(((Yc -
1)/50)*ylim)))
                    elec_density(Xc,Yc) = elec_density(Xc,Yc) + 1;
     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

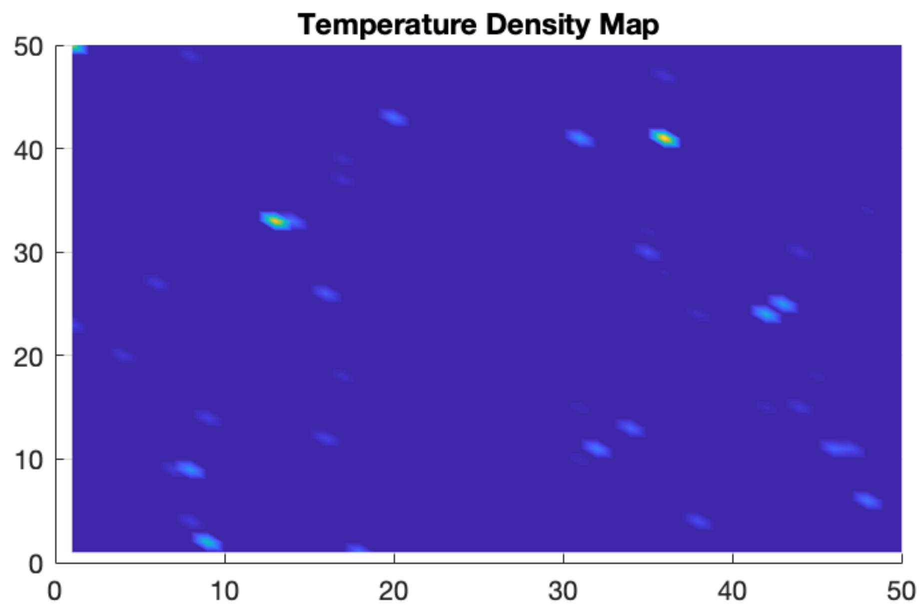


Temperature and electron density maps were then developed for the regions. The plots can be seen below. The noteable result is that there are no traces of electrons in the boxed regions. The temperature density is related to the random scatter and distribution still occuring in the region, hence the non-uniformity in the points.

```
ELEC_density = elec_density';

figure(2)
[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

Temp_density = temperature_density';
```

```
figure(3)
[A, B] = meshgrid(1:1:50,1:1:50);
surf(Temp_density)
shading interp
title('Temperature Density Map');
view(0,90);
```





Published with MATLAB® R2017a