Assignment 3: Part 3

The Finite Difference method was then used to calculate the the potential within a region with a bottle neck inserted. The same approach was taken in assignment 2 part 2. As well, the bottle necked regions were applied in the form of rectangles that the particles could no longer travel through. This program was created in Assignment 1 part 3 and modified in the code below. The simluation results can be found below.

```
close all;
clear;

Sigma = 1;

nx = 50;
ny = 50;

G = sparse (nx*ny, nx*ny);
B = zeros(1, nx*ny);

cMap = zeros (nx, ny);

%Loop to assign Conductivity
for i = 1:nx
    for j = 1:ny
        if ((i>=0.4*nx) && (i<=0.6*nx) && (j<=0.4*ny)) || ((i>=0.4*nx)
 && (i<=0.6*nx) && (j>=0.6*ny))
            cMap(i,j) = .01;
        else
            cMap(i,j) = Sigma;
        end
    end
end

for i = 1:nx
    for j = 1:ny
        n = j + (i-1)*ny;

        if i == 1
            G(n, :) = 0;
            G(n, n) = 1;
            B(n)    = 1;
        elseif i == nx
            G(n, :) = 0;
            G(n, n) = 1;
        elseif j == 1
            nxm = j + (i - 2)*ny;
            nxp = j + (i)*ny;
            nyp = j + 1 + (i - 1)*ny;

            rxm = (cMap(i,j) + cMap(i - 1, j))/2;
            rxp = (cMap(i,j) + cMap(i + 1, j))/2;
            ryp = (cMap(i,j) + cMap(i, j + 1))/2;
```

```
            G(n, n) = -(rxm + rxp + ryp);
            G(n, nxm) = rxm;
            G(n, nxp) = rxp;
            G(n, nyp) = ryp;

        elseif j == ny
            nxm = j + (i - 2)*ny;
            nxp = j + (i)*ny;
            nym = j - 1 + (i - 1)*ny;

            rxm = (cMap(i,j) + cMap(i - 1, j))/2;
            rxp = (cMap(i,j) + cMap(i + 1, j))/2;
            rym = (cMap(i,j) + cMap(i, j - 1))/2;

            G(n, n) = -(rxm + rxp + rym);
            G(n, nxm) = rxm;
            G(n, nxp) = rxp;
            G(n, nym) = rym;

        else

            nxm = j + (i - 2)*ny;
            nxp = j + (i)*ny;
            nym = j - 1 + (i - 1)*ny;
            nyp = j + 1 + (i - 1)*ny;

            rxm = (cMap(i,j) + cMap(i - 1, j))/2;
            rxp = (cMap(i,j) + cMap(i + 1, j))/2;
            rym = (cMap(i,j) + cMap(i, j - 1))/2;
            ryp = (cMap(i,j) + cMap(i, j + 1))/2;

            G(n, n) = -(rxm + rxp + ryp + rym);
            G(n, nxm) = rxm;
            G(n, nxp) = rxp;
            G(n, nym) = rym;
            G(n, nyp) = ryp;

        end

    end

end

V = G\B';

vMap = zeros(nx,ny);
for i=1:nx
    for j=1:ny
        n = j + (i - 1)*ny;
        vMap(i,j) = V(n);
    end
end

vMap_T = vMap';
```

```
[Ex, Ey] = gradient(-vMap_T);
EX = Ex/5E-8;
EY = Ey/5E-8;
```

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);
q = 1.602E-19;

%region limits
xlim = 50E-9;
ylim = 50E-9;
x_low_lim = 0.4*xlim;
x_high_lim = 0.6*xlim;
y_low_lim = 0.4*ylim;
y_high_lim = 0.6*ylim;

force_x = EX * q;
force_y = EY * q;
a_x = force_x/mn;
a_y = force_y/mn;

voltage = 0.8;
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.

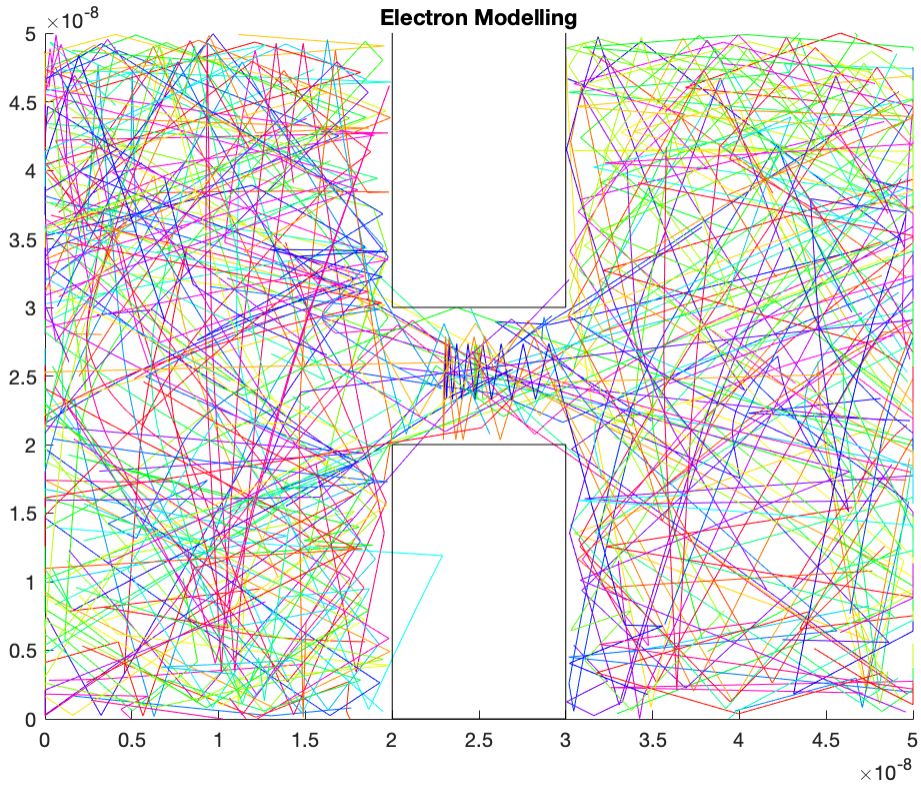
```
num_particles = 100;
%initialize all of the particles
% 1 - x
% 2 - y
% 3 - direction (angle)
% 4 - vth
% 5 - vx
% 6 - vy

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 * 50;
```

```
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));
        end

        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: 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
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

        if (particle_vector(n, 1)<0)
            particle_vector(n, 1) = 0;
        end


        this = particle_vector(n,6)*time_step;
        particle_vector(n,2) + particle_vector(n,6)*time_step;
        %y boundary conditions - particle reflects at the same angle
        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));
        end

        next_x = (particle_vector(n,1) +
 particle_vector(n,5)*time_step);
        next_y = (particle_vector(n,2) +
 particle_vector(n,6)*time_step);

        %x low case
        if (particle_vector(n, 1) < x_low_lim && ((next_x) >
 x_low_lim) && ((next_y >= y_high_lim)||(next_y <= y_low_lim)))
                particle_vector(n, 5) = - particle_vector(n, 5);
%         x high case
        elseif (next_x <= x_high_lim && particle_vector(n, 1) >
 x_high_lim && ((next_y >= y_high_lim)||( next_y <= y_low_lim)))
                particle_vector(n, 5) = - particle_vector(n, 5);
        %y high case
        elseif (next_y >= y_high_lim && next_x >= x_low_lim && next_x
 <= x_high_lim)
                particle_vector(n, 6) = - particle_vector(n, 6);
%        %y low case
        elseif ((next_y <= y_low_lim) && (next_x >= x_low_lim &&
 next_x <= x_high_lim && particle_vector(n, 2) >=(y_low_lim)))
            particle_vector(n, 6) = - particle_vector(n, 6);
        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]);
            rectangle('Position',[x_low_lim 0 (x_high_lim - x_low_lim)
 y_low_lim]);
            rectangle('Position',[x_low_lim y_high_lim (x_high_lim -
 x_low_lim) 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);

    %account for the electric field
    for i = 1:(num_particles)
        position_x = particle_vector(i, 1)*(1E9);
        position_y = particle_vector(i, 2)*(1E9);
        x_pos = round(position_x);
```

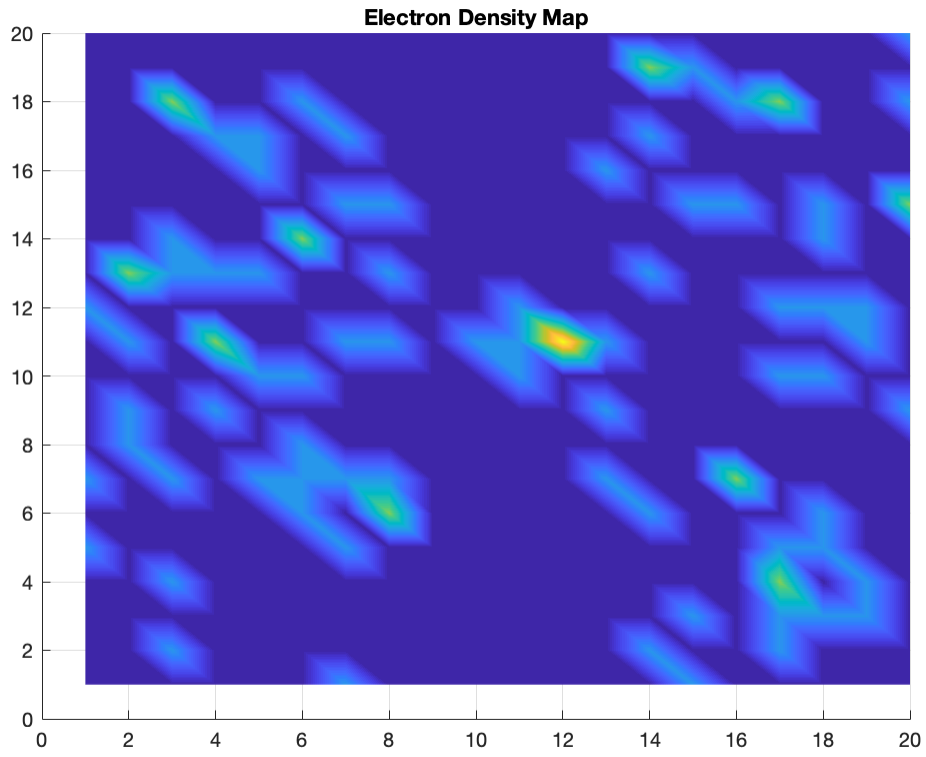
```
            y_pos = round(position_y);

            if (y_pos == 0)
                y_pos = 1;
            elseif (y_pos < 0)
                y_pos = -1 * y_pos;
            end

            if (y_pos > ylim*1E9)
                y_pos = ylim*1E9;
            end

            if (x_pos == 0)
                x_pos = 1;
            elseif (x_pos < 0)
                x_pos = -1 * x_pos;
            end

            if(x_pos > xlim*1E9)
                x_pos = xlim*1E9;
            end

            particle_vector(i,5) = particle_vector(i,5) + a_x(x_pos,
 y_pos)*time_step;
            particle_vector(i,6) = particle_vector(i,6) + a_y(x_pos,
 y_pos)*time_step;
        end

        particle_vector(:,1) = particle_vector(:,1) + particle_vector(:,
 5).*time_step;
        particle_vector(:,2) = particle_vector(:,2) + particle_vector(:,
 6).*time_step;

        title 'Electron Modelling';
        hold on;
        pause(0.01);

        time = time + time_step;
        time_array(l, 1) = time;

    end
```



A temperature density map was then created using the same methods used in part 1 of the assignment. One may note that, at the end of the simulation, the majority of electrons are on the left hand side of the blocks. This is not suprising given the electric field created. If one refers back to part 2 of the lab where the potential map was created, one side is on a much high potential. It is therefore logical that the electrons would clutter around that side. The electron density map can be found below. Note that if the simulation were to continue for a longer duration, the results would be even more prominent.

```
elec_density = zeros(20,20);
sizeparticle = size(particle_vector);

for Xc=1:20
    for Yc = 1:20
        for count = 1:sizeparticle
           if((particle_vector(count,
 1)<=((Xc/20)*xlim))&&(particle_vector(count, 1)>(((Xc -
 1)/20)*xlim)))
                if((particle_vector(count,
 2)<=((Yc/20)*ylim))&&(particle_vector(count, 2)>(((Yc -
 1)/20)*ylim)))
                    elec_density(Xc,Yc) = elec_density(Xc,Yc) + 1;
                end
           end
        end
    end
end
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);
```



This assignemnt monitored particles movement and position when in a constant electric field defined by a bottle neck. The particles movement was based on this electric field. However, this could improved to be more realistic by considering each particle as a charge. Using this analagy, each particle would effect eachothers motion which would lead to a much more realistic simulation.

Published with MATLAB® R2017a