ELEC 4700 Assignment 3: Monte-Carlo/Finite Difference Method

Table of Contents

Jacob Godin - 100969991

Part 1

a) If a voltage of 0.1V is applied across the x dimension of the semiconductor, what is the electric field on the electrons?

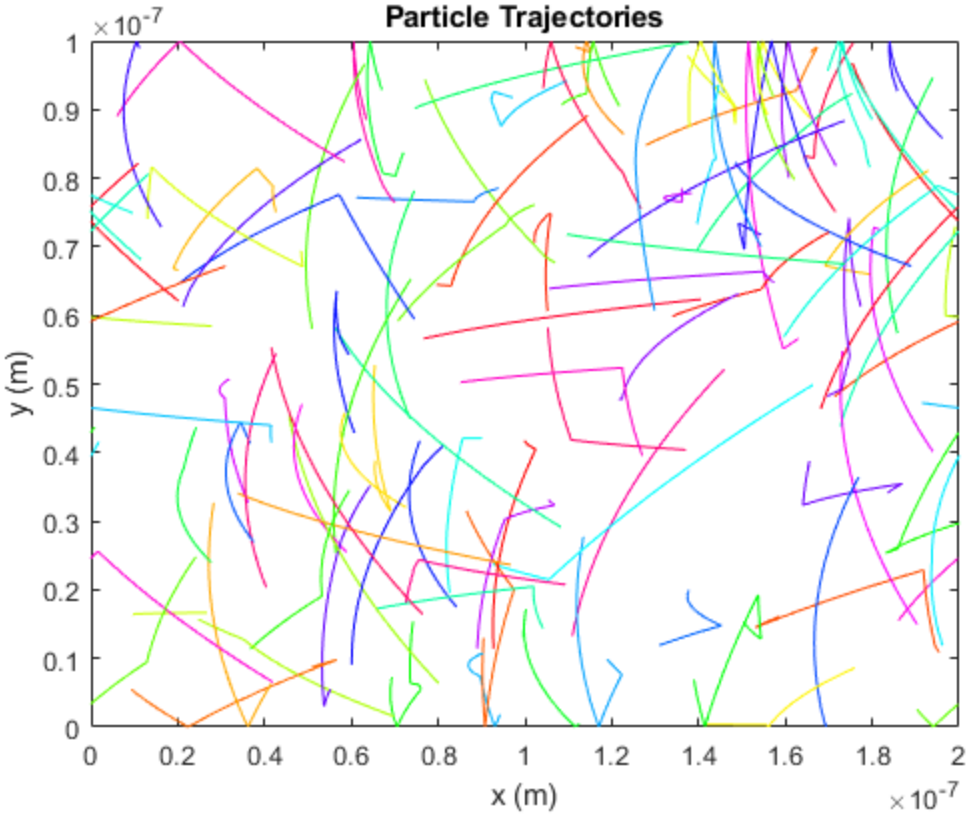
```
clear; close all; clc;

global mn, global k, global T;

V = 0.3;
m0 = 9.11e-31;
mn = 0.26*m0;
dim_x = 200e-9;
dim_y = 100e-9;
k = 1.38064852e-23;
T = 300;
q = 1.602e-19;

% Assuming that the electric field is constant through the sample,
 applying
% a 0.1 V potential across the x-dimension will yield an electric
 field of
% E = dV/dx

E = V/dim_x;
fprintf("The electric field on the electrons is %d V/m\n", E);

% b) What is the force on each electron?

F = q*E;
fprintf("The force on each electron is %d N\n", F);

% c) Calculate the acceleration on the electrons and use it in the
 model to
% update the velocity of each electron at each time step.

acceleration = F/mn;
```

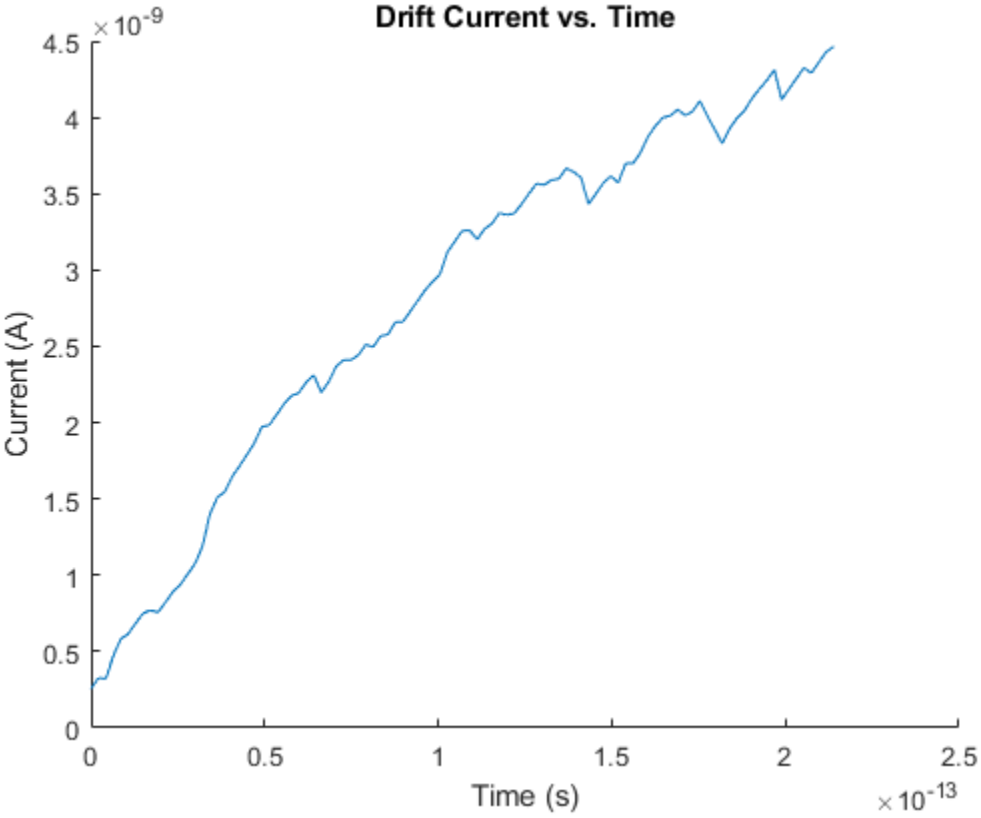
```
fprintf("The acceleration of each electron is %d m/s^2.\n",
 acceleration);

% The acceleration due to a force acting on the electron can be
 calculated
% by using Newton's Law of motion, where F = m*a.

% Thermal Velocity
Vth = sqrt(2*k*T/mn);

% Mean free path
Tmn = 0.2e-12;
Mfp = Tmn * Vth;

num_e = 100;

% Calculate number of electrons in system
density = 10.^19; %1/m^2
E_density = density*dim_x*dim_y;

% initialize x and y position of electrons
[x_vec, y_vec] = initPosition(num_e, dim_x, dim_y);

% initialize x and y velocity of electrons
[vx_vec, vy_vec] = initVelocity(num_e, Vth);

% initialize time variables
t = 0;
steps = 100;
t_step = max(dim_x, dim_y)/(500*Vth);
t_final = steps*t_step;
t_vec = zeros(1,steps+1);

% initialize time vector with time step
t_vec = zeros(1,length(t_vec));
for i=1:length(t_vec)
    t_vec(i) = (i-1)*t_step;
end

colour = hsv(num_e);
j=0;
Temp = zeros(1,length(t_vec));
avgXSpeed = zeros(1,length(t_vec));
avgSpeed = zeros(1,length(t_vec));

% Scattering probability
P_scatter = 1 - exp(-t_step/Tmn);

while t < t_final
    j=j+1;
    % Calculate temp
    Temp(j) = (mean(vx_vec.^2 + vy_vec.^2)*mn)/(2*k);

    % Calculate new velocity if electron scatters
```

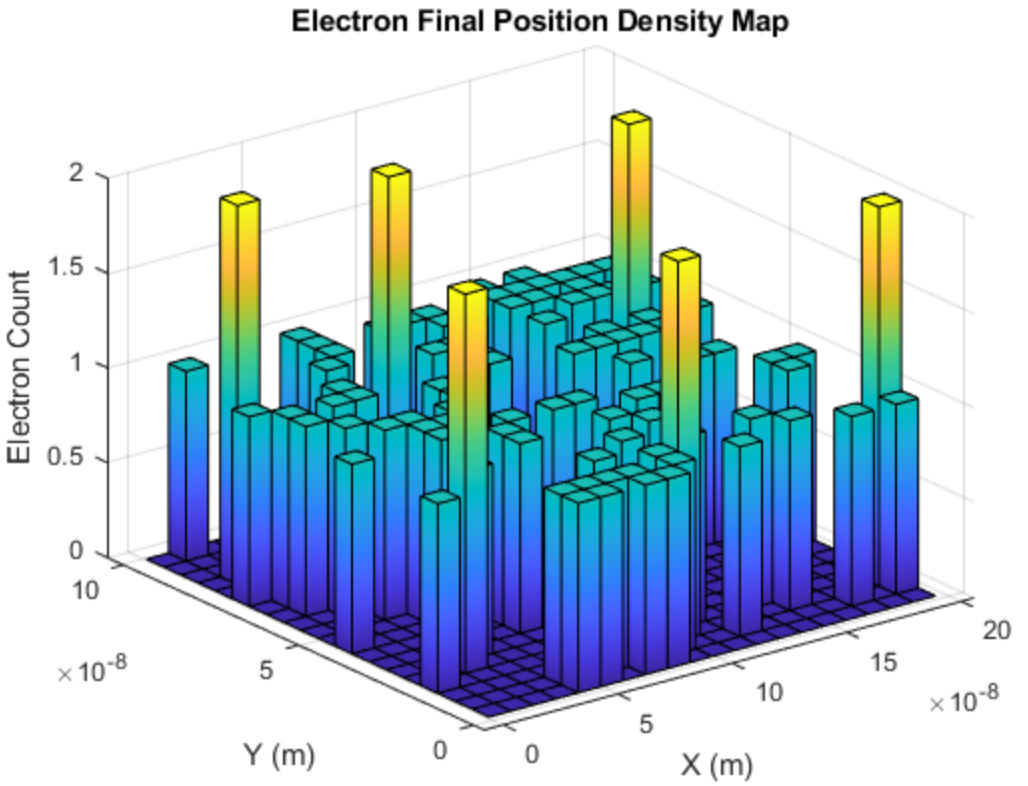
```
    for i=1:length(x_vec)
        if P_scatter > rand()
            [vx_vec(i), vy_vec(i)] = newBoltDist();
            vx_vec(i) = vx_vec(i) + acceleration*t_step;
        end
    end

    % Update x-velocity with acceleration
    vx_vec = vx_vec + acceleration*t_step;

    % Save previous positions
    x_vec_prev = x_vec;
    y_vec_prev = y_vec;

    % Calculate new position
    x_vec = x_vec + vx_vec*t_step;
    y_vec = y_vec + vy_vec*t_step;

    % Boundary conditions
    for i=1:num_e
        if x_vec(i) < 0 % left boundary, periodic
            x_vec(i) = x_vec(i)+dim_x;
            x_vec_prev(i) = dim_x;
        end
        if x_vec(i) > dim_x % right boundary, periodic
            x_vec(i) = x_vec(i)-dim_x;
            x_vec_prev(i) = 0;
        end
        if y_vec(i) > dim_y % top boundary, reflect
            vy_vec(i) = -vy_vec(i);
            y_vec(i) = 2*dim_y - y_vec(i);
        end
        if y_vec(i) < 0 % bottom boundary, reflect
            vy_vec(i) = -vy_vec(i);
            y_vec(i) = abs(y_vec(i));
        end
    end

    % Plot trajectories
    figure(1);
    xlabel('x (m)')
    ylabel('y (m)')
    title('Particle Trajectories')
    xlim([0 dim_x])
    ylim([0 dim_y])
    pause(0.1)
    % Figure 1: A simulated 2D electron trajectory of electrons
travelling
    % through a silicon sample with an applied electric field about
the
    % x-dimension. The electric potential was increased to 0.3V to
    % exaggerate the curvature of the electrons' trajectories. The
    % curvature is only present in the electrons' x-velocity
component.
```

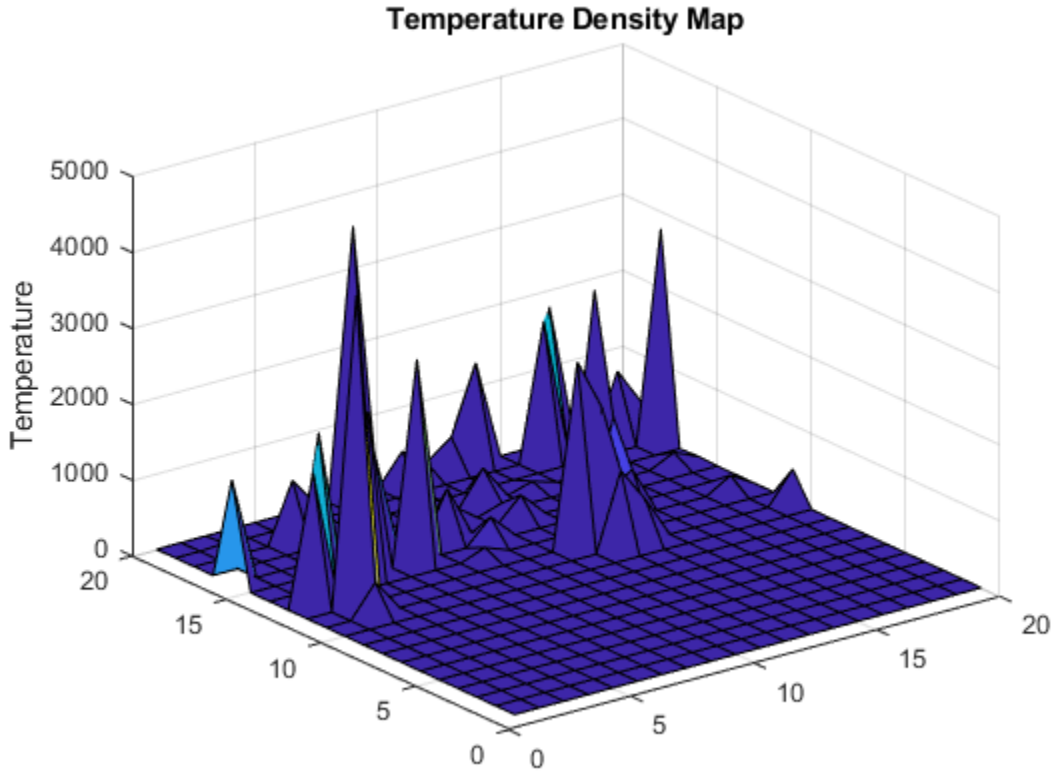
```
    for i=1:num_e
        plot([x_vec_prev(i);x_vec(i)],
[y_vec_prev(i);y_vec(i)],'color',colour(i,:))
        hold on
    end

    % X-speed over time
    avgXSpeed(j) = mean(vx_vec);

    % Calculate temperature
    Vsq = vx_vec.^2 + vy_vec.^2;
    Temp(j) = (mean(Vsq)*mn)/(2*k);

    t=t+t_step;
end

% Plot drif current over time
current_x = q*E_density*avgXSpeed;
% The current is calculated by the product of the fundamental charge,
 q,
% the electron density, and the average velocity of the electron in
 the x
% dimension.
figure(2);
xlabel("Time (s)");
ylabel("Current (A)");
title("Drift Current vs. Time");
hold on;
plot(t_vec,current_x)

% Electron final position density map
Z = [transpose(x_vec), transpose(y_vec)];
figure(3);
hist3(Z, [20,20]);
hold on;
xlabel('X (m)');
ylabel('Y (m)');
zlabel('Electron Count');
title('Electron Final Position Density Map');
surfHandle = get(gca, 'child');
set(surfHandle,'FaceColor','interp', 'CdataMode', 'auto');

% Temperature map
[binx,biny]=meshgrid(0:dim_y/20:dim_y,0:dim_x/20:dim_x);
temp=zeros(20,20);
counter=0;
vtotal=0;
for i=1:20
    txmn=binx(1,i);
    txmx=binx(1,i+1);
    for r =1:20
        tymn=biny(r,1);
        tymx=biny(r+1,1);
```

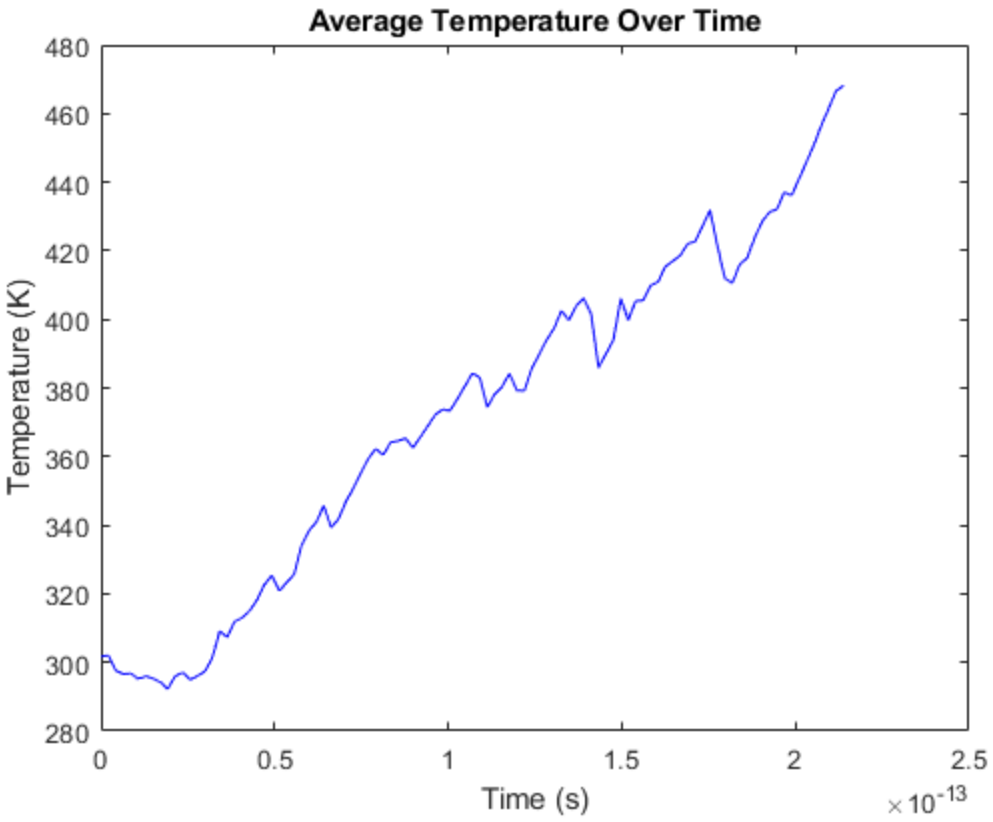
```
        for mm=1:num_e
            if(x_vec(mm)>txmn & x_vec(mm)<txmx & y_vec(mm)<tymx &
 y_vec(mm)>tymn)
                counter=counter+1;
                vtotal=vtotal+sqrt(vx_vec(mm)^2+vy_vec(mm)^2);
                if(counter~=0)
                    temp(r,i)=mn*(vtotal^2)/(counter*k);
                end
            end
        end
        vtotal=0;
        counter=0;
    end
end
%Question 3: Plot the electron density map
figure(4)
surf(flipud(temp))
title('Temperature Density Map')
zlabel('Temperature')

% Plot temperature over time
figure(5);
plot(t_vec,Temp, 'b')
xlabel('Time (s)');
ylabel('Temperature (K)');
title('Average Temperature Over Time');

The electric field on the electrons is 1500000 V/m
The force on each electron is 2.403000e-13 N
The acceleration of each electron is 1014523347124883840 m/s^2.
```


Particle Trajectories
×10^-7
y (m)
x (m)
×10^-7
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8
1.2
1.4
1.6
1.8
2


Drift Current vs. Time
×10^-9
Current (A)
Time (s)
×10^-13
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


Electron Final Position Density Map
Electron Count
2
1.5
1
0.5
0
10
×10^-8
5
0
Y (m)
0
5
10
15
20
×10^-8
X (m)


Temperature Density Map
Temperature
5000
4000
3000
2000
1000
0
20
15
10
5
0
0
5
10
15
20



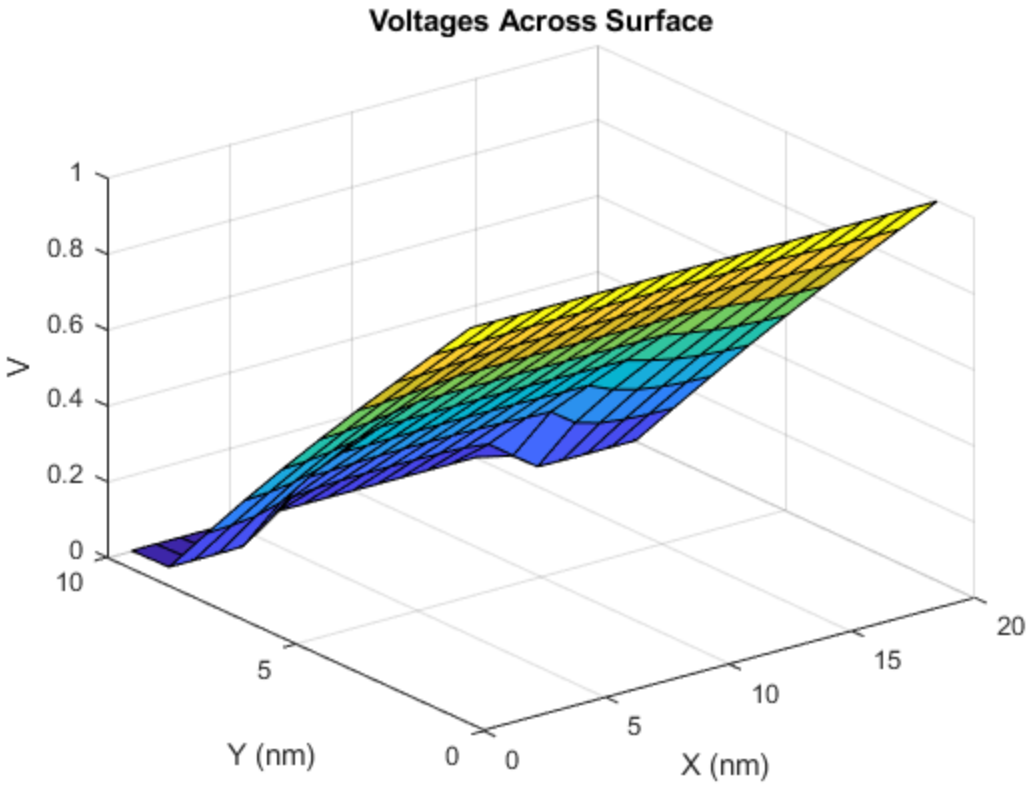
Part 2

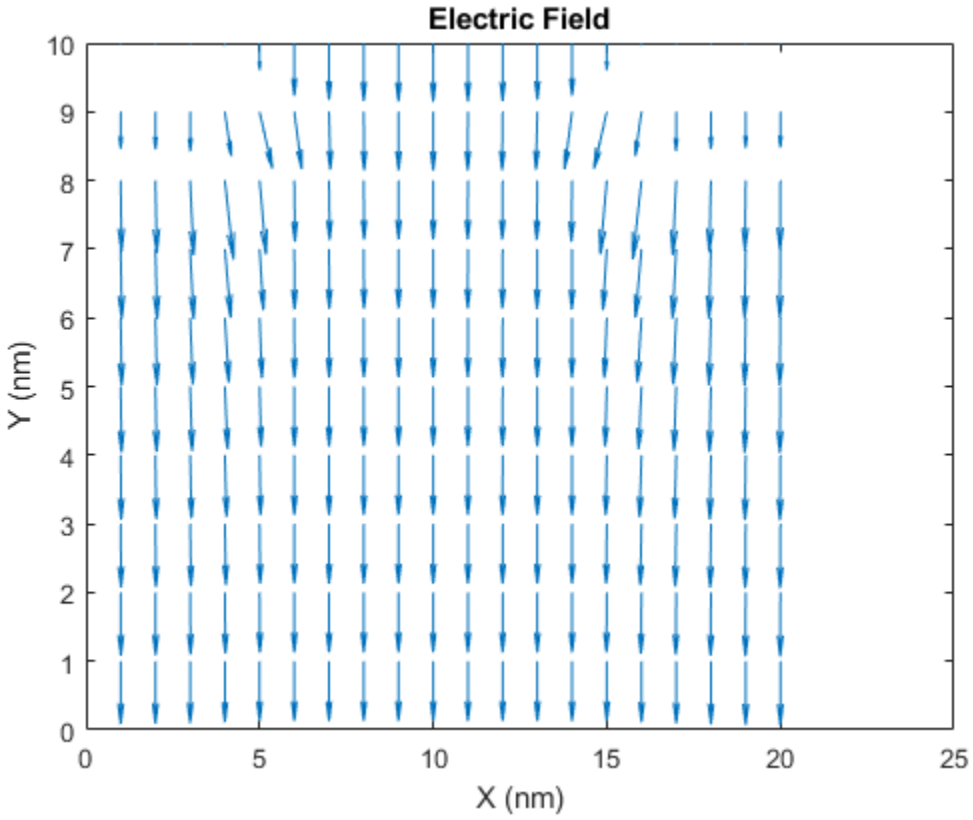
Use the Finite Difference Method in Assignment-2 to calculate the electric field.

```
% a) Use the code from Assignment-2 to calculate the potential with
 the
% bottle-neck inserted.

% Inputs
W = 20;
L = 0.5*W;

nx = L;
ny = W;

V0 = 1;

% Surface map
sMap = zeros(nx,ny);

sig1 = 1;
sig2 = 1e-2;

Wb = 6;
Lb = 10;
```

```
top_box = [(nx/2)-(Lb/2) 0 Lb Wb];
bottom_box = [(nx/2)-(Lb/2) ny-Wb Lb Wb];

% Populate sigma matrix
sigma = ones(nx,ny);
for i=1:nx
    for j=1:ny
        if (i > top_box(1) && i < top_box(1)+top_box(3) && (j <
 bottom_box(4) || j > bottom_box(2)))
            sigma(i,j) = sig2;
        end
    end
end

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

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

        if i == 1 % Left
            G(n,:) = 0;
            G(n,n) = 1;

            B(n) = V0;
        elseif i == nx % Right
            G(n,:) = 0;
            G(n,n) = 1;

            B(n,1) = 0;
        elseif j == 1 % Bottom
            nxm = j + (i-2)*ny;
            nxp = j +(i)*ny;
            nyp = j+1 + (i-1)*ny;

            rxm = (sigma(i,j) + sigma(i-1,j))./2.0;
            rxp = (sigma(i,j) + sigma(i+1,j))./2.0;
            ryp = (sigma(i,j) + sigma(i,j+1))./2.0;

            G(n,n) = -(rxm+rxp+ryp);
   G(n,nxm) = rxm;
   G(n,nxp) = rxp;
   G(n,nyp) = ryp;
        elseif j == ny % Top
            nxm = j + (i-2)*ny;
            nxp = j +(i)*ny;
            nym = j-1 + (i-1)*ny;

            rxm = (sigma(i,j) + sigma(i-1,j))./2.0;
            rxp = (sigma(i,j) + sigma(i+1,j))./2.0;
            rym = (sigma(i,j) + sigma(i,j-1))./2.0;
```

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

        else % Not along any boundary
            nxm = j + (i-2)*ny;
            nxp = j +(i)*ny;
            nym = j-1 + (i-1)*ny;
            nyp = j+1 + (i-1)*ny;

            rxm = (sigma(i,j) + sigma(i-1,j))./2.0;
            rxp = (sigma(i,j) + sigma(i+1,j))./2.0;
            rym = (sigma(i,j) + sigma(i,j-1))./2.0;
            ryp = (sigma(i,j) + sigma(i,j+1))./2.0;

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

phi_vec = G\B;

for i=1:nx
    for j=1:ny
        n = j + (i-1)*ny;
        sMap(i,j) = phi_vec(n);
    end
end

figure(6)
surf(sMap)
title('Voltages Across Surface')
zlabel('V');
xlabel('X (nm)');
ylabel('Y (nm)');

% Electric Field
[Ex, Ey] = gradient(sMap);

figure(7)
quiver(Ex,Ey)
title('Electric Field')
xlabel('X (nm)');
ylabel('Y (nm)');
```


Voltages Across Surface
V
1
0.8
0.6
0.4
0.2
0
10
5
0
Y (nm)
0
5
10
15
20
X (nm)


Electric Field
Y (nm)
10
9
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
X (nm)

Part 3

Use the Finite Difference Method to provide a feild for the Monte-Carlo bottle-neck simulation.

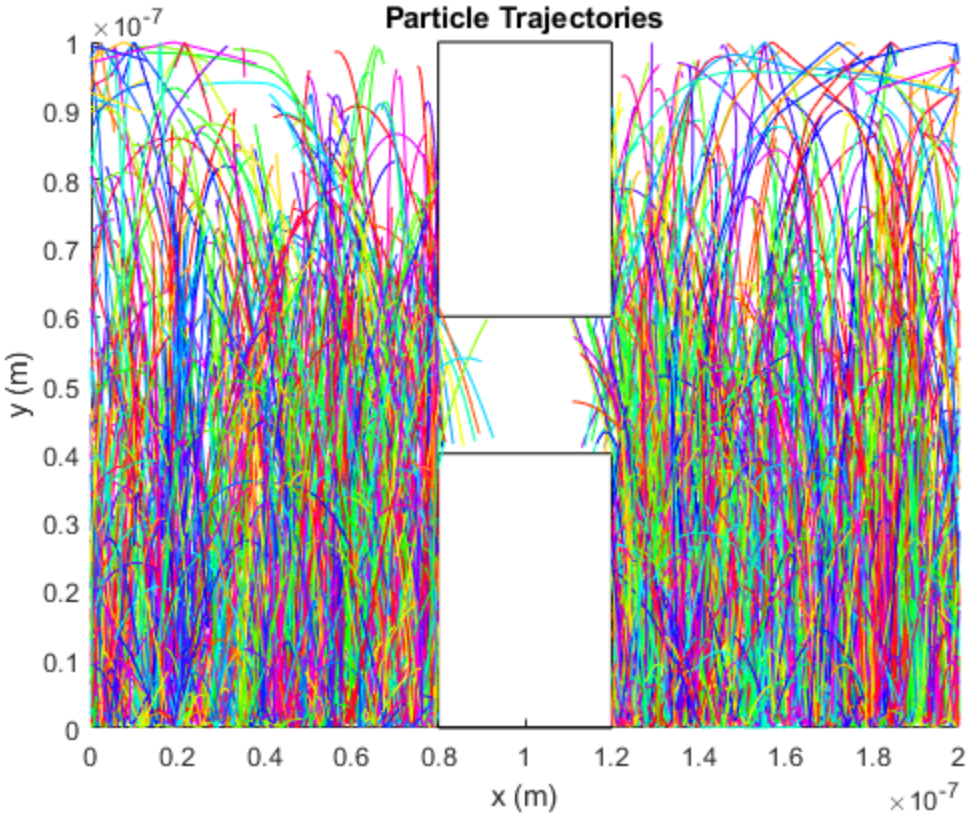
```
m0 = 9.11e-31;
mn = 0.26*m0;
dim_x = 200e-9;
dim_y = 100e-9;
k = 1.38064852e-23;
T = 300;
q = 1.602e-19;
num_e = 1000;

% Define box limits
box_x = 40;
box_y = 40;

% Box limits no greater than 120x40
if box_x > 120
    boy_x = 120;
end

if box_y > 40
    box_y = 40;
end

box_x = box_x*1e-9;
box_y = box_y*1e-9;

lim_x_low = (dim_x/2)-(box_x/2);
lim_x_high = (dim_x/2)+(box_x/2);

% Draw boxes on figure
figure(8);
hold on;
rectangle('position', [lim_x_low 0 box_x box_y]);
rectangle('position', [lim_x_low dim_y-box_y box_x box_y]);
axis([0 dim_x 0 dim_y])

% Initialize x and y positions of electrons
[x_vec, y_vec] = initPositionOutsideBox(num_e, dim_x, dim_y, box_x,
 box_y);

% Initialize x and y velocities
[vx_vec, vy_vec] = initBoltDist(num_e);

t = 0;
mode = 1; %1 for specular, 0 for diffusive boundaries

Vth = sqrt(2*k*T/mn);
Tmn = 0.2e-12;
Mfp = Tmn * Vth;
t_step = max(dim_x, dim_y)/(250*Vth);
```

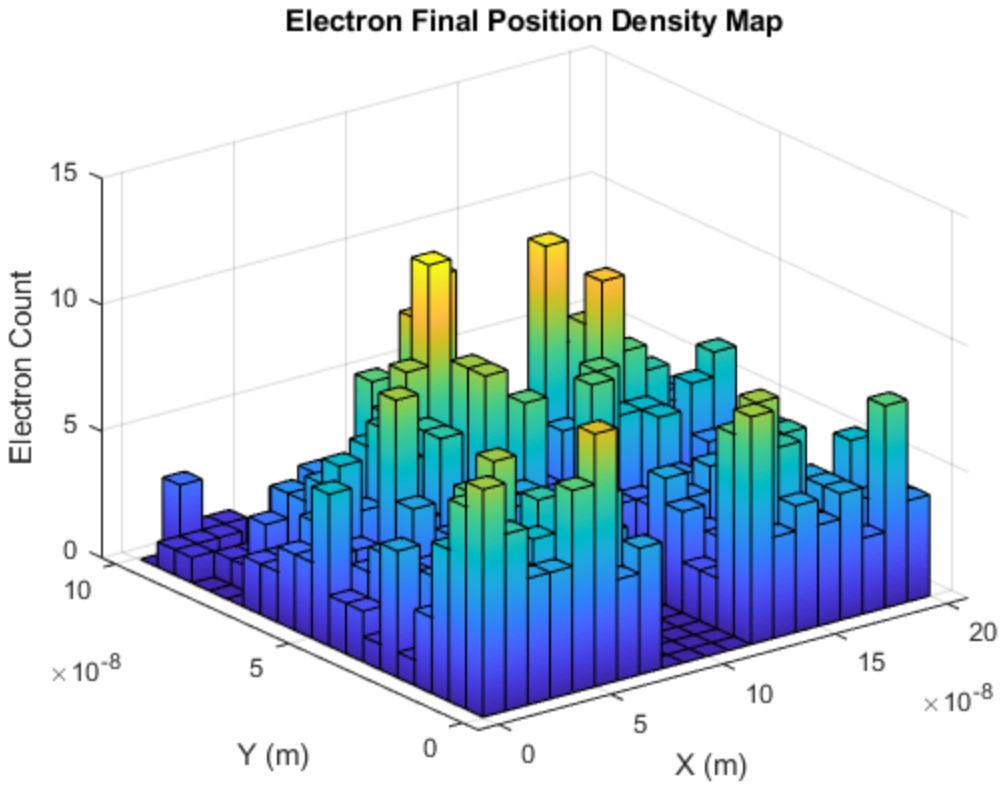
```
steps = 50;
t_final = steps*t_step;

P_scatter = 1 - exp(-t_step/Tmn);

accelerationX = zeros(num_e,1);
accelerationY = zeros(num_e,1);

Fx = q.*Ex;
Fy = q.*Ey;
accelerationX = Fx./mn;
accelerationY = Fy./mn;

colour = hsv(num_e);
index=zeros(num_e,1);

while t < t_final

    roundvarray=floor(x_vec*10^8)+1;
    roundharray=floor(y_vec*10^8)+1;

    index=sub2ind(size(Ex),[roundharray],[roundvarray]);

    accelerationX = 10^8*Ex(index)*q/mn;
    accelerationY = 10^8*Ey(index)*q/mn;

    % Calculate new velocity if electron scatters
%     for i=1:length(x_vec)
%         if P_scatter > rand()
%             [vx_vec(i), vy_vec(i)] = newBoltDist();
%         end
%     end

    % Update x and y velocity with acceleration
    vx_vec = vx_vec + accelerationX.*t_step;
    vy_vec = vy_vec + accelerationY.*t_step;

    % Save previous positions
    x_vec_prev = x_vec;
    y_vec_prev = y_vec;

    % Calculate new position
    x_vec = x_vec + vx_vec*t_step;
    y_vec = y_vec + vy_vec*t_step;

    for i=1:num_e
        if mode == 1
            % Boundary conditions for semiconductor edges
            if x_vec(i) < 0 % left boundary, periodic
                x_vec(i) = x_vec(i)+dim_x;
                x_vec_prev(i) = dim_x;
            end
            if x_vec(i) > dim_x % right boundary, periodic
                x_vec(i) = x_vec(i)-dim_x;
```

```
                x_vec_prev(i) = 0;
            end
            if y_vec(i) > dim_y % top boundary, reflect
                vy_vec(i) = -vy_vec(i);
                y_vec(i) = 2*dim_y - y_vec(i);
            end
            if y_vec(i) < 0 % bottom boundary, reflect
                vy_vec(i) = -vy_vec(i);
                y_vec(i) = abs(y_vec(i));
            end
            % Boundary conditions for box edges
            % if x_vec(i) hits left or right edge of box between box y
            % boundaries, reflect
            if (x_vec(i) > lim_x_low && x_vec(i) < lim_x_high) &&
~(y_vec(i) > box_y && y_vec(i) < (dim_y-box_y))
                vx_vec(i) = -vx_vec(i);

                % Remove particles from box
                x_vec(i) = x_vec_prev(i);
                y_vec(i) = y_vec_prev(i);
            end
            % if y_vec(i) hits bottom or top edge of box between box x
            % boundaries, reflect
            if (y_vec(i) < box_y && y_vec(i) > dim_y-box_y) &&
(x_vec(i) > lim_x_low && x_vec(i) < lim_x_high)
                vy_vec(i) = -vy_vec(i);

                % Remove particles from box
                x_vec(i) = x_vec_prev(i);
                y_vec(i) = y_vec_prev(i);
            end

        elseif mode == 0
            % Boundary conditions for semiconductor edges
            if x_vec(i) < 0 % left boundary, periodic
                x_vec(i) = x_vec(i)+dim_x;
                x_vec_prev(i) = dim_x;
            end
            if x_vec(i) > dim_x % right boundary, periodic
                x_vec(i) = x_vec(i)-dim_x;
                x_vec_prev(i) = 0;
            end
            if y_vec(i) > dim_y % top boundary, reflect
                vy_vec(i) = -vy_vec(i);
                y_vec(i) = 2*dim_y - y_vec(i);
            end
            if y_vec(i) < 0 % bottom boundary, reflect
                vy_vec(i) = -vy_vec(i);
                y_vec(i) = abs(y_vec(i));
            end
            % Boundary conditions for box edges
            % if x_vec(i) hits left or right edge of box between box y
            % boundaries, reflect
```

```
            if (x_vec(i) > lim_x_low && x_vec(i) < lim_x_high) &&
 ~(y_vec(i) > box_y && y_vec(i) < (dim_y-box_y))
                vy_vec(i) = newBoltDist();
                vx_vec(i) = newBoltDist();

                % Remove particles from box
                x_vec(i) = x_vec_prev(i);
                y_vec(i) = y_vec_prev(i);
            end
            % if y_vec(i) hits bottom or top edge of box between box x
            % boundaries, reflect
            if (y_vec(i) < box_y && y_vec(i) > dim_y-box_y) &&
 (x_vec(i) > lim_x_low && x_vec(i) < lim_x_high)
                vy_vec(i) = newBoltDist();
                vx_vec(i) = newBoltDist();

                % Remove particles from box
                x_vec(i) = x_vec_prev(i);
                y_vec(i) = y_vec_prev(i);
            end
        end
    end


    % Plot trajectories
    xlabel('x (m)')
    ylabel('y (m)')
    title('Particle Trajectories')
    pause(0.1)

    for i=1:num_e
        plot([x_vec_prev(i);x_vec(i)],
[y_vec_prev(i);y_vec(i)],'color',colour(i,:))
        hold on
    end

    t=t+t_step;
end

% Electron final position density map
Z = [transpose(x_vec), transpose(y_vec)];
figure(9);
hist3(Z, [20,20]);
hold on;
xlabel('X (m)');
ylabel('Y (m)');
zlabel('Electron Count');
title('Electron Final Position Density Map');
surfHandle = get(gca, 'child');
set(surfHandle,'FaceColor','interp', 'CdataMode', 'auto');

% b) The electric field inside the bottle neck is relatively higher
 than
% outside the bottleneck. This means that the electrons that are
```

```
% travelling through the bottleneck are accelerated at quicker rate
 than
% from outside the bottleneck.

% c) To make this simulation more accurate, a third space dimension
 could
% be added. This would allow for a more realistic represenation of the
% density and temperature maps.
```





Published with MATLAB® R2017b