ELEC 4700 Assignment 3 Monte-Carlo/Finite Difference Method

Student: Samuel (Wendi) Zhu

Student Number: 101088968

Date: 3/19/2022

Contents

Question 1

Start with the Monte-Carlo simulator from Assignment-1 without the bottle-neck:

```
% This file is the code for ELEC 4700 assignment 1 question 2
% Electron Modeling

% Clear all
clearvars
```

```
clearvars -global
close all
format shorte

% Make plot pretier
% set(0,'DefaultFigureWindowStyle','docked')
% set(0,'defaultaxesfontsize',20)
set(0,'defaultaxesfontname','Times New Roman')
set(0,'DefaultLineLineWidth', 2);

% Global variables
global C        % constants module that holds all the constants
global x y      % arrays for the current electrons positions: 1 row, colum for current position
global xp yp    % arrays for the previous electrons positions: 1 row, column for previous
position
global vx vy    % arrays for current electrons velocities: 1 row, column for current velocity
global ax ay    % scalars for electron acceleration in x and y direction
global limits   % Limits for the plot
% Initalize global constants
% Electron charge
C.q_0 = 1.60217653e-19;  % C
% Rest mass
C.m0 = 9.1093837015e-31; % KG
% Effective mass of electrons
C.mn = 0.26*C.m0;  % KG
% Boltzmann constant
C.kb = 1.38064852e-23;  %m^2 * kg * s^-2 * K^-1

% Initialize the region size 200nm X 100nm
Region.x = 200e-9;
Region.y = 100e-9;
limits = [0 Region.x 0 Region.y];  % plot limit
% Initialize the temperature
T = 300;  % K
vth = sqrt(2*C.kb*T/C.mn);  % Calculate the thermal velocity
% Initialize the mean time between collision
Tmn = 0.2e-12;  % 0.2ps
d = Tmn*vth;
fprintf("Expected Mean free path is "+ d + " m\n");
% Initialize the number of "super" electrons
numE = 10000;
numEPlot = 5;  % Number of electron to be plotted
% Initialize the time
deltaT = 2e-14; % Time interval per simulation step in second
pauseTime = 0.02; % Time paused per simulation step in second
simTime = 0;  % Hold the current simulation time
% Number of simulation steps
numSim = 1000;
% Temperature grid
numGridX = 15; % number of grid in x direction
numGridY = 15; % number of grid in y direction
% Array to hold temperature over time
tempOverTime = zeros(1,numSim);
% Variables for actual mean free paths and mean collision time calculations
```

```
totalFP = 0;  % total free path
totalFT = 0;  % total free time
countFPFT = 0;  % count for scattering
arrScatterPx = zeros(1, numE);  % Hold the previous scattering point: index to target previous
point for an electron
arrScatterPy = zeros(1, numE);
arrScatterT = zeros(1, numE);  % Hold the previous scattering time: index to target previous
scatter time for an electron

% Voltage across the x dimension
voltageX0 = 0;  % V; Voltage at x = 0, the left side of the region
voltageX1 = 0.1;   % V; Voltage at x = L, the right side of the region
```

```
Expected Mean free path is 3.7404e-08 m
```

Q1 a)

If a voltage of 0.1V is applied across the x dimension of the semiconductor, the electric field on the electrons can be calculated using E = -(V0-V1)/(x0-x1). Note that the 0.1 V is applied at the right side of the region.

$$\boldsymbol{E} = -\frac{0-0.1}{0-200\times10^{-9}} = -500000\frac{V}{m}\hat{x}$$

```
EfieldX = - (voltageX0-voltageX1)/(0-Region.x);  % V/m; Calculate the Electric field in x
direction
% Print the electric field calculation result
fprintf("The voltage at the left side is: V(x=0) = "+voltageX0+" V\n")
fprintf("The voltage at the right side is: V(x=0) = "+voltageX1+" V\n")
fprintf("The electric field is: "+EfieldX+" V/m in x-direction\n")
```

```
The voltage at the left side is: V(x=0) = 0 V
The voltage at the right side is: V(x=0) = 0.1 V
The electric field is: -500000 V/m in x-direction
```

Q1 b)

The force on each electron can be calculated using F = qE.

$$\boldsymbol{F} = -1.60217653\times10^{-19}\times-500000 = 8.0109\times10^{-14}\,N\,\hat{x}$$

```
forceEx = -C.q_0 * EfieldX;  % N; calculate the force on each electron in x direction
% Print the force calculation result
fprintf("The force on each electron is: "+forceEx + " N in x-direction\n")
```

```
The force on each electron is: 8.0109e-14 N in x-direction
```

Q1 c)

The acceleration of the electrons can be calculated using a = F/m.

$$\boldsymbol{a} = \frac{8.0109\times10^{-14}}{0.26\times9.1093837015\times10^{-31}} = 3.3823\times10^{17}\,m/s^2\,\hat{x}$$

```
% Calculate the acceleration due to static electric field
ax = forceEx/C.mn;  % Calculate the electron acceleration in x direction
ay = 0;  % Calculate the electron acceleration in y direction
% Print the acceleration calculation result
fprintf("Electron acceleration in x direction is: "+ax+" m/s^2\n")
fprintf("Electron acceleration in y direction is: "+ay+" m/s^2\n")
```

```
Electron acceleration in x direction is: 3.382345955008032e+17 m/s^2
Electron acceleration in y direction is: 0 m/s^2
```

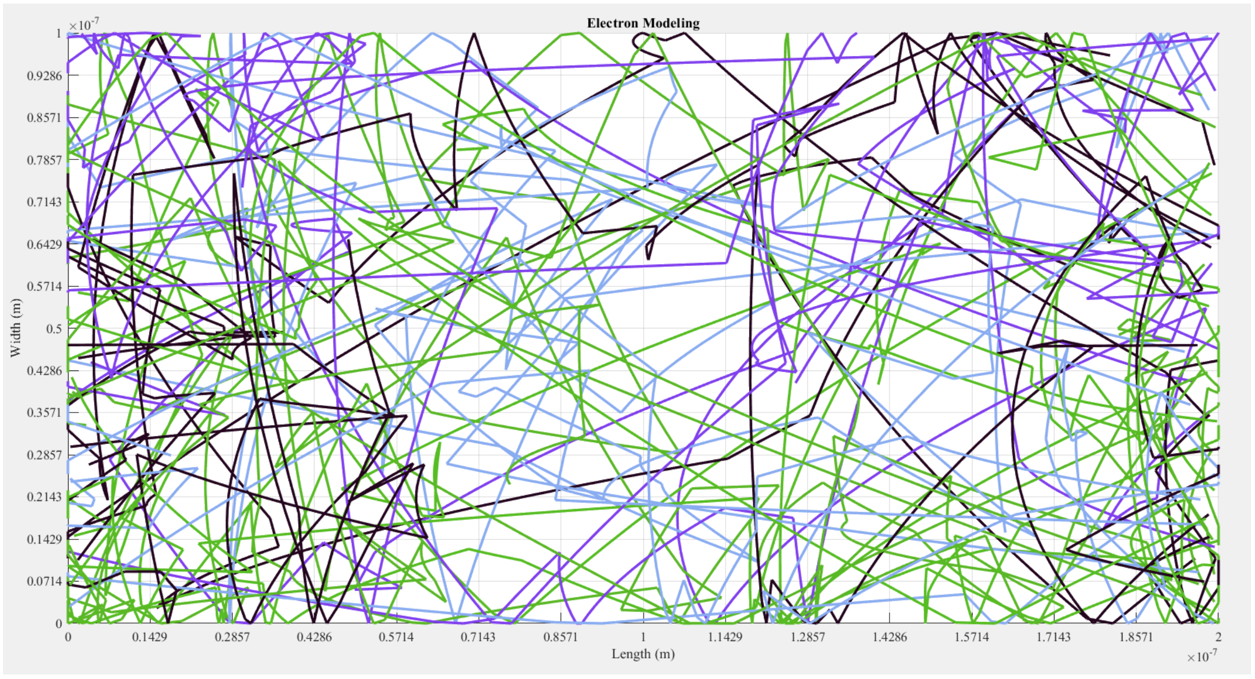

Figure 1 2D plot of particle trajectories

Q1 d)

The electron drift current density is related to the average carrier velocity using the formula J = n*v*q, where n is the electron density [1/m^2] , v is the velocity [m/s], q is charge [C]. To find the current, the following steps are performed:

```
% 1) Calculate the total area
areaA = Region.x * Region.y;  % m^2
areaA = areaA * 100^2;  % cm^2
% 2) Calculate the total electrons in the area assuming electron
% concentration is 10^15 cm-2
totalE = 10^15 * areaA;  % total electrons
% 3) Find the charge per "Super Electron", where "Super Electron" is the
% particle in this simulation
superECharge = -C.q_0 * totalE/numE;  % Charge per super electron
% 4) The current can be found by counting the net number of super electrons
% flow through the left side over the deltaT.
vectorCurrent = zeros(1, numSim);  % Create a vector to hold the current over simulation
```

```
vectorTime = deltaT*(1:numSim);  % Time vector for the simulation use for plot


% Add the electrons
AddElectrons(numE, Region, vth, T);

% Calculate the scattering probability
Pscat = 1-exp(-deltaT/Tmn);

% Initalize plot
figure(2)
axCol = axes;
axCol.ColorOrder = rand(numEPlot,3); % Initalize color for each electron
hold on

% Loop for simulation
for iSim = 1:numSim
     PlotPoint(numEPlot, numGridX, numGridY);

     % Store the current positions
     xp = x;
     yp = y;
     % Calculate the future positions: x = x0 + vx*t
     x = x + vx * deltaT;
     y = y + vy * deltaT;
     % Calculate the future velocity: vx = ax*t
     vx = vx + ax*deltaT;
     vy = vy + ay*deltaT;

     % Increment simulation time
     simTime = simTime + deltaT;

     % Super electron count for current calculation
     % Count on left side x=0. +1 flow right, -1 flow left
     countECurrent = 0;  % Hold the super electron count

    % Loop through all the particles
    for iE=1:numE
        % flag for invalid
        bInvalid = false;
        % Check for invalid x position
        if x(iE) < 0
            x(iE) = Region.x; % Appear on right
            xp(iE) = x(iE);
            bInvalid = true;
            % Update the electron count for current calculation
            countECurrent = countECurrent-1;  % -1 flow left
        elseif x(iE) > Region.x
            x(iE) = 0; % Appear on left
            xp(iE) = x(iE);
            bInvalid = true;
            % Update the electron count for current calculation
            countECurrent = countECurrent+1;  % +1 flow right
        end
        % Check for invalid y position
        if y(iE) < 0
            y(iE) = 0; % Reflect
            vy(iE) = -vy(iE);
            bInvalid = true;
        elseif y(iE) > Region.y
            y(iE) = Region.y; % Reflect
            vy(iE) = -vy(iE);
            bInvalid = true;
        end

        % Check for scattering
        if ~bInvalid && Pscat > rand()
            % Rethermalize
            vx(iE) = sqrt(C.kb*T/C.mn).*randn();
            vy(iE) = sqrt(C.kb*T/C.mn).*randn();
            % Calculate the free path
            deltaX = x(iE) - arrScatterPx(iE);
            deltaY = y(iE) - arrScatterPy(iE);
            totalFP = totalFP + sqrt(deltaX^2 + deltaY^2);
            arrScatterPx(iE) = x(iE);  % Update the previous scatter position
            arrScatterPy(iE) = y(iE);
            % Calculate the free time
            totalFT = totalFT + simTime - arrScatterT(iE);
            arrScatterT(iE) = simTime;  % Update the previous scatter time
            % Increment the count
            countFPFT = countFPFT+1;
        end
    end

    % Calculate the average temperature
    vth2_mean = mean(sqrt(vx.^2+vy.^2)).^2;
    tempOverTime(iSim) = C.mn*vth2_mean/(2*C.kb);

    % Calculate the current
    vectorCurrent(iSim) = superECharge*countECurrent/deltaT;

    % Pause some time
    pause(pauseTime);
end

% Plot the current over time
figure(3)
plot(vectorTime, abs(vectorCurrent));
title("Current over time");
xlabel("Time (s)")
ylabel("Current magnitude (A)")
grid on
snapnow
```

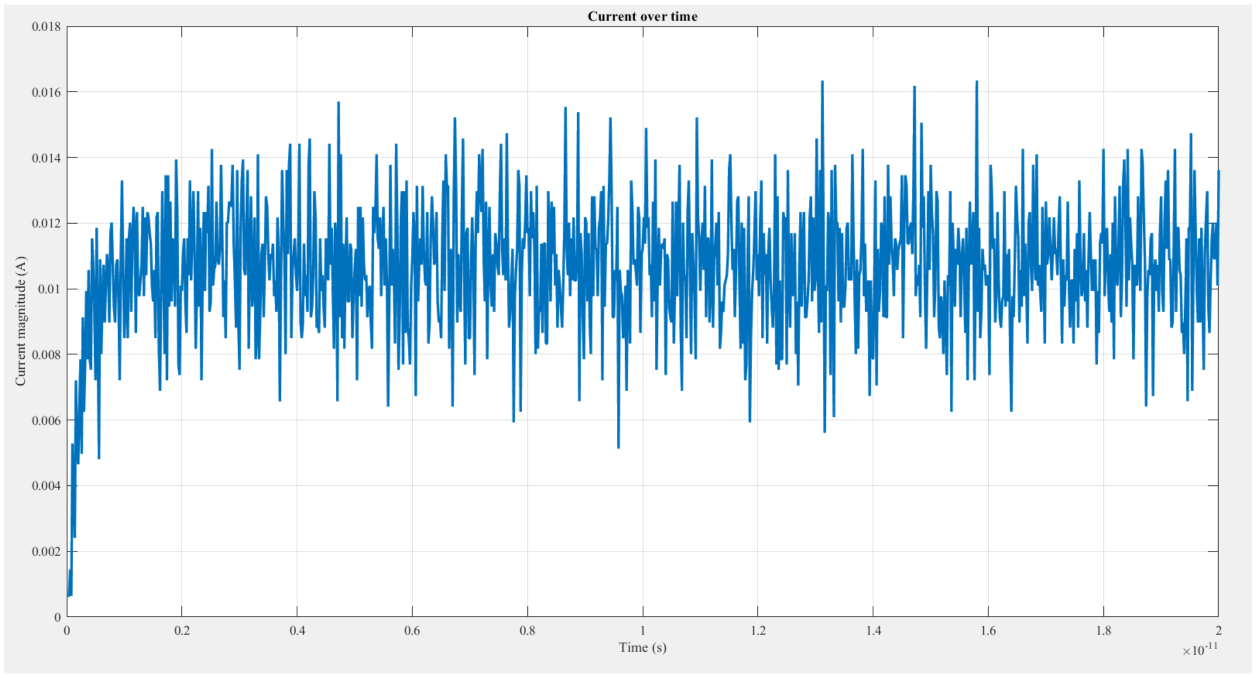

Figure 2 Current behavior over time

Q1 d Comment)

The magnitude of the current is increasing at the beginning since the electrons are accelerating. After some time, the magnitude of the current become relatively constant because the acceleration due to E field and the scattering effect cancel out. The direction of the current is defined in the direction of the x-axis. The current flow in this case is negative because the "net electrons" are flowing along the x-axis, and the charge of the electron is negative. This make sense since we apply 0 volts on the left side and a positive voltage on the right side, so the current is expected to flow from right to left, which is negative in the direction of the x-axis.

Q1 e) Density plot and temperature plot

```
% Plot the temperature and density plot
tempDisplay(numGridX, numGridY, numE, Region.x, Region.y);

% Plot average temperature over time
figure(6)
plot(vectorTime, tempOverTime);
title("Temperature over time");
xlabel("Time (s)");
ylabel("Temperature (K)");
ylim([0 inf]);
grid on
snapnow

% Calculate the actual mean free path and mean time between collision
meanFreePath = totalFP/countFPFT;
meanTimeCollision = totalFT/countFPFT;
```

```
fprintf("Actual mean free path: "+meanFreePath + " m\n");
fprintf("Mean time between collision: "+meanTimeCollision + " s\n");
```

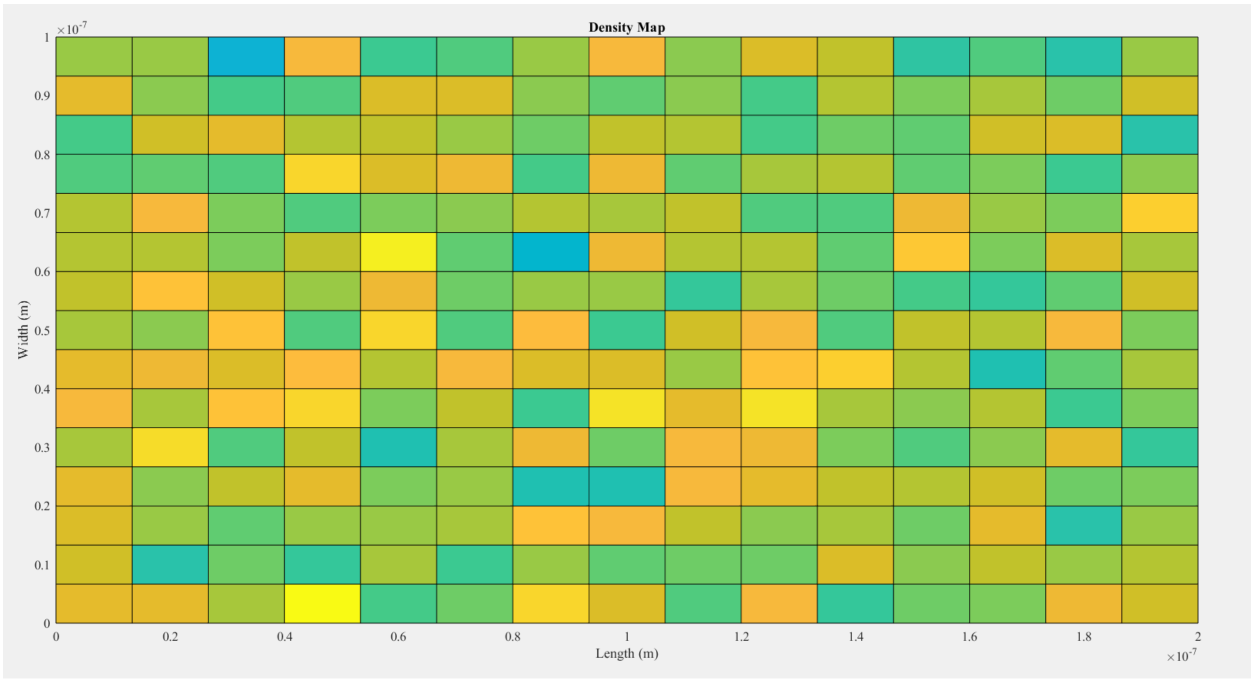

Figure 3 Plot of density map

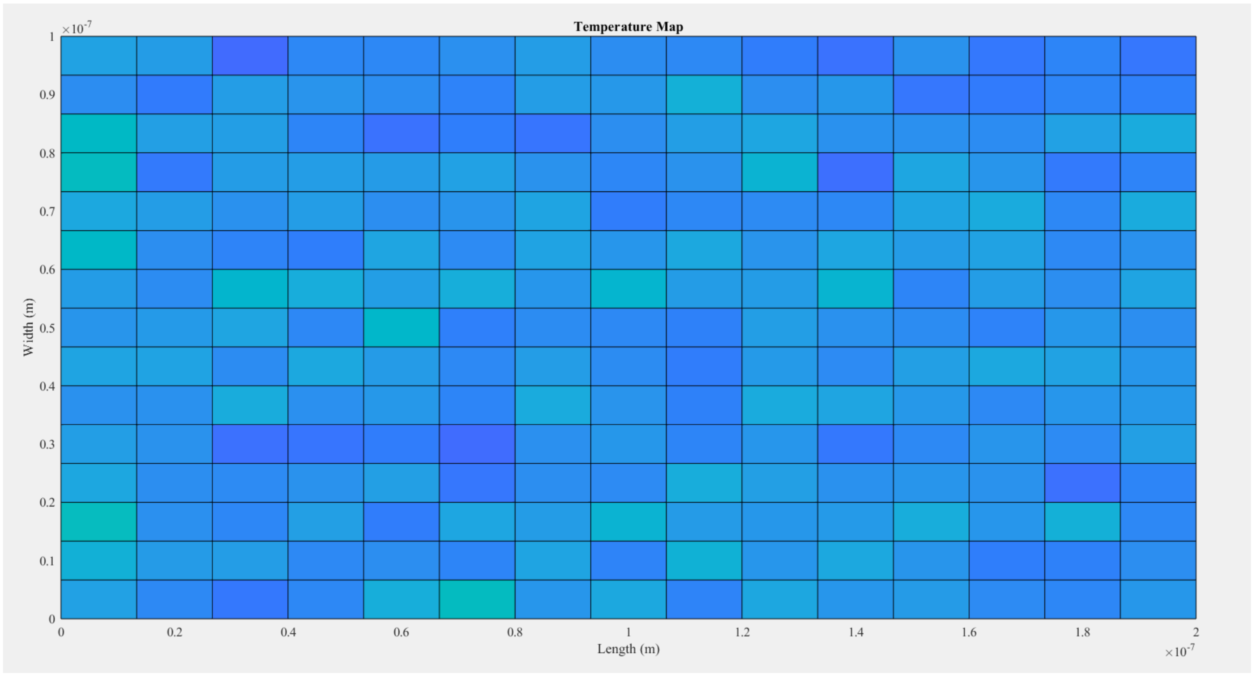

Figure 4 Plot of temperature map

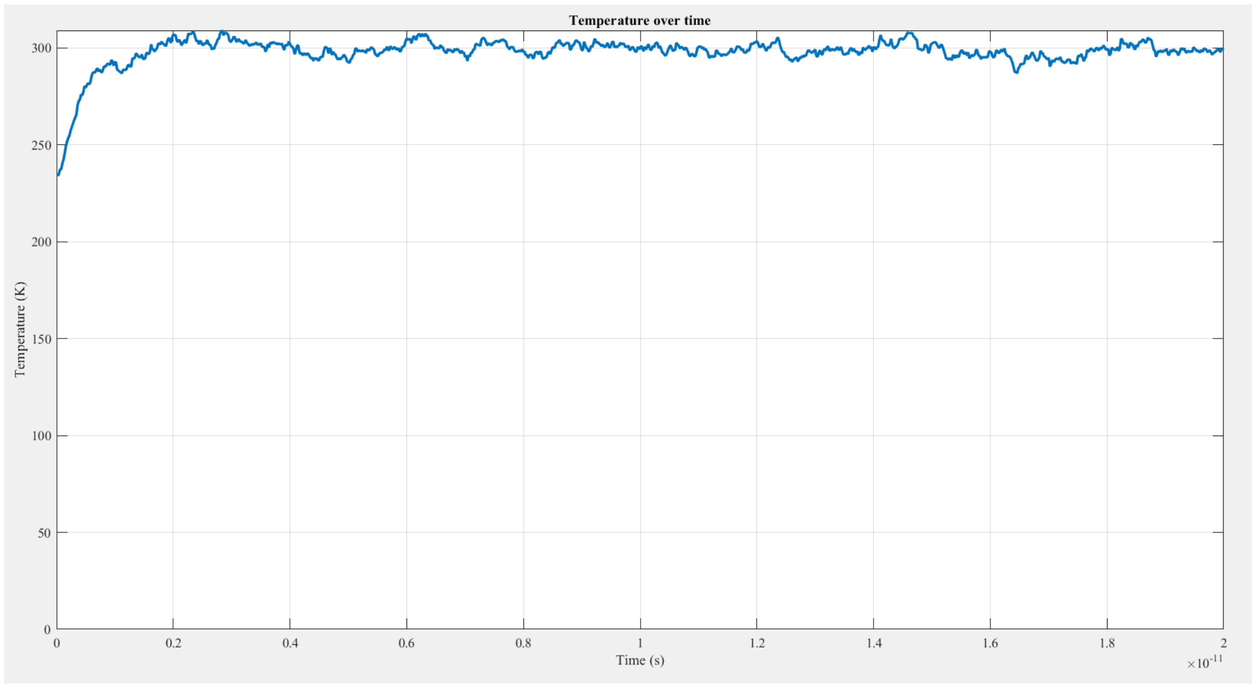

Figure 5 Plot of temperature over time

```
Actual mean free path: 3.8671e-08 m
Mean time between collision: 2.151e-13 s
```

Question 2

Use the code from Assignment-2 to calculate the potential with the bottle-neck inserted.

```
% Define the dimension
L = Region.x * 10^9;  % Length in nm
W = Region.y * 10^9;  % Width in nm
boxLF = 0.3;  % Fraction of the length of the box
boxWF = 0.4;  % Fraction of the width of the box
Lb = boxLF*L;  % Length of the box in nm
Wb = boxWF*W;  % Width of the box in nm
deltaXY = 0.02*L;  % Assume deltaX = deltaY in nm

% Calculate the dimension of solution matrix
nx = (L/deltaXY);
ny = (W/deltaXY);
[X,Y] = meshgrid(linspace(0,L,nx), linspace(0,W,ny));

% Declare the matrix for conductivity: Sigma(y,x)
matrixSigma = ones(ny, nx);  % Dimension: ny times nx
xIndexBox = ceil((L-Lb)/(2*deltaXY));  % Find the starting x index for the box
LbIndexRange = ceil(Lb/deltaXY);  % Index range for the length of the box
WbIndexRange = ceil(Wb/deltaXY);  % Index range for the width of the box
% Assign the region for the box
matrixSigma(1:WbIndexRange, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;
matrixSigma(ny-WbIndexRange:ny, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;
```

```
% Plot the region for conductivity
figure(7)
surf(X,Y, matrixSigma)
title("Plot of Conductivity Sigma(x,y)")
xlabel("X axis - Length (nm)")
ylabel("Y axis - Width (nm)")
zlabel("Z axis - Conductivity (S)")
view(0,90)  % View from top
snapnow

% Declare the matrix for voltage V(y,x)
matrixV = zeros(ny, nx);  % Dimension: ny times nx

% Declare the G matrix and F vector: GV = F
G = zeros(nx*ny, nx*ny);
F = zeros(nx*ny, 1);

% Construct the G matrix and F vector
for ix = 1:nx
    for iy = 1:ny
        % Calculate the index
        n = mappingEq(ix, iy, ny);
        % Check for the boundary
        if ix==1 || ix==nx || iy ==1 || iy==ny
            G(n,n) = 1;
            % Boundary condition for x
            if ix == 1
                F(n,1) = voltageX0;  % V at x = 0
            elseif ix == nx
                F(n,1) = voltageX1;  % and V at x = L
            elseif iy == 1
                nyp = mappingEq(ix, iy+1, ny);  % dV/dy=0 at y=0
                G(n,nyp) = -1;
            elseif iy == ny
                nym = mappingEq(ix, iy-1, ny);  % dV/dy=0 at y=W
                G(n, nym) = -1;
            end
        else
            % Calculate the sigma
            sigmaxp = (matrixSigma(iy,ix) + matrixSigma(iy,ix+1))/2;
            sigmaxm = (matrixSigma(iy,ix) + matrixSigma(iy, ix-1))/2;
            sigmayp = (matrixSigma(iy,ix) + matrixSigma(iy+1, ix))/2;
            sigmaym = (matrixSigma(iy,ix) + matrixSigma(iy-1, ix))/2;

            % Calculate mapping index
            nxp = mappingEq(ix+1, iy, ny);  % index for V(i+1,j)
            nxm = mappingEq(ix-1, iy, ny);  % index for V(i-1,j)
            nyp = mappingEq(ix, iy+1, ny);  % index for V(i,j+1)
            nym = mappingEq(ix, iy-1, ny);  % index for V(i,j-1)

            % Setup the G matrix
            G(n,n) = -(sigmaxp+sigmaxm+sigmayp+sigmaym)/deltaXY^2;
            G(n, nxp) = sigmaxp/deltaXY^2;
            G(n, nxm) = sigmaxm/deltaXY^2;
            G(n, nyp) = sigmayp/deltaXY^2;
            G(n, nym) = sigmaym/deltaXY^2;
        end
    end
end

% Solve for V from GV = F
V = G\F;

% Map back to the 2D region
for iMap = 1:nx*ny
    % Calculate the index for the 2D region
    ix = ceil(iMap/ny);
    iy = mod(iMap, ny);
    if iy == 0
        iy = ny;
    end
    % Assign the value
    matrixV(iy, ix) = V(iMap);
end
```

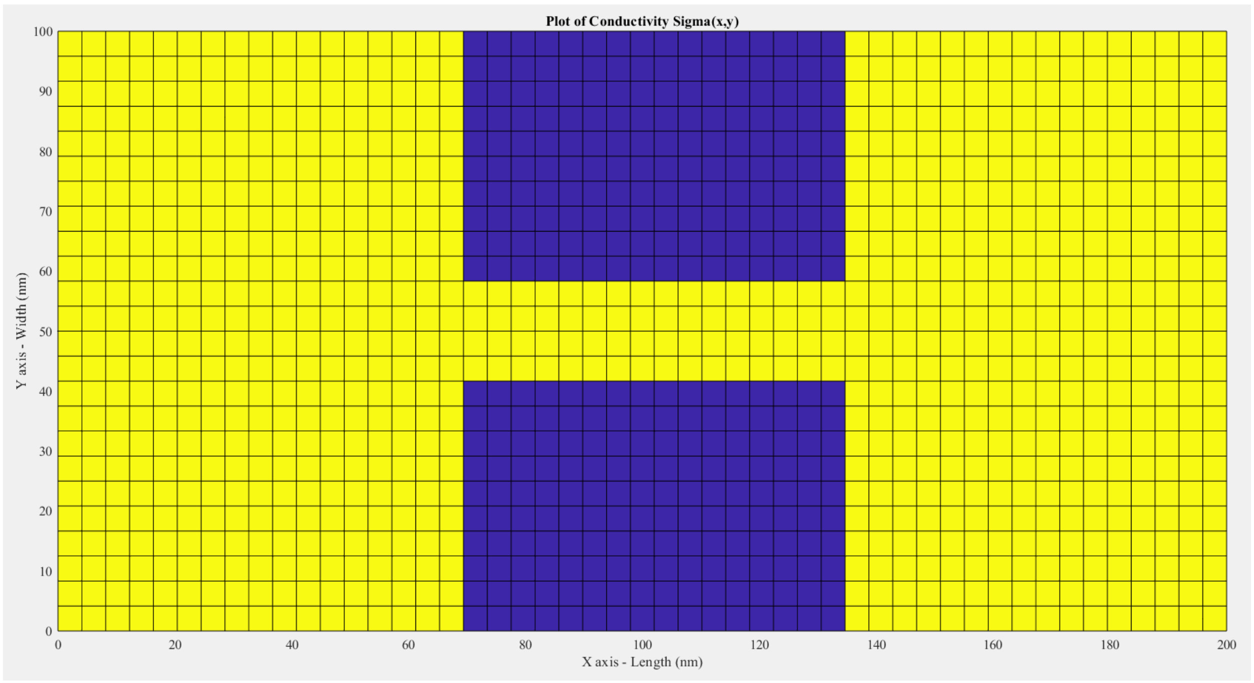

Figure 6 Plot of conductivity

Q2 a)

Plot the solution for V from the Finite Difference Method

```
figure(8)
surf(X,Y,matrixV)
xlabel("X axis - Length (nm)")
ylabel("Y axis - Width (nm)")
```

```
zlabel("Z axis - Voltage (V)")
title("Surface Plot of Voltage V(x,y)")
% view(45,45)  % 3-D View
snapnow
```

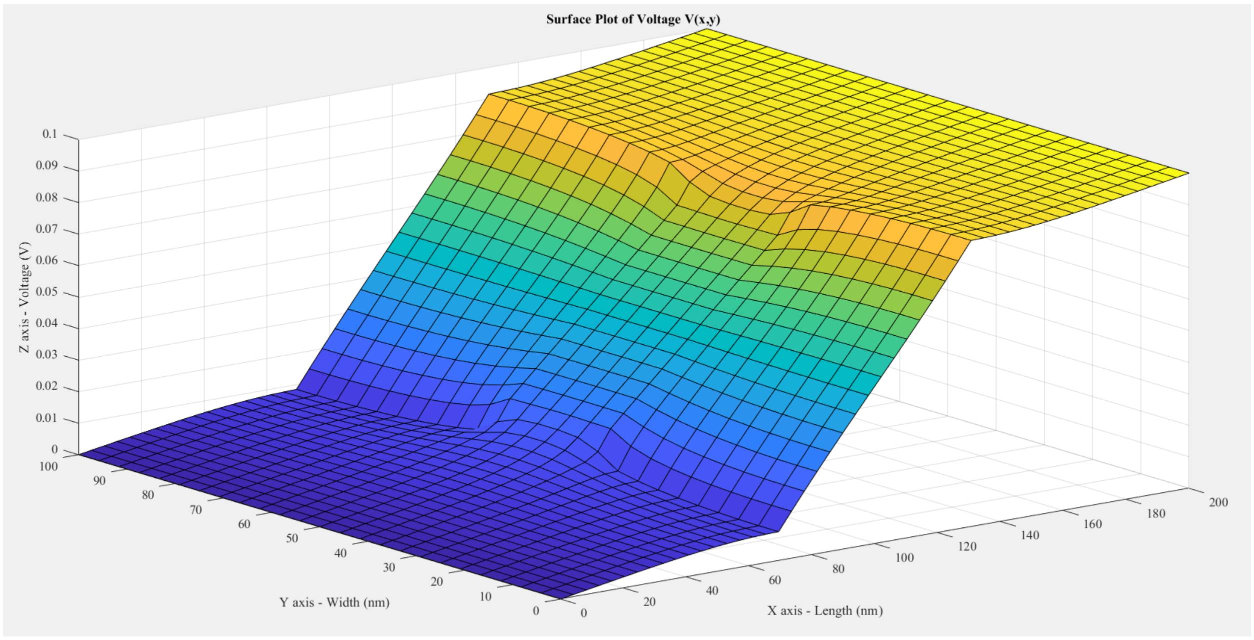

Figure 7 Surface plot of voltage potential

Q2 b)

Solve the electric field

```
[Ex, Ey] = gradient(-matrixV);
Ex = Ex/(deltaXY * 10^-9);  % convert to V/m
Ey = Ey/(deltaXY * 10^-9);  % convert to V/m
% Plot the electric field
figure(9)
quiver(X, Y, Ex, Ey);
title("Plot of Electric Field E(x,y) (V/m)")
xlabel("X axis - Length (nm)")
ylabel("Y axis - Width (nm)")
snapnow
```

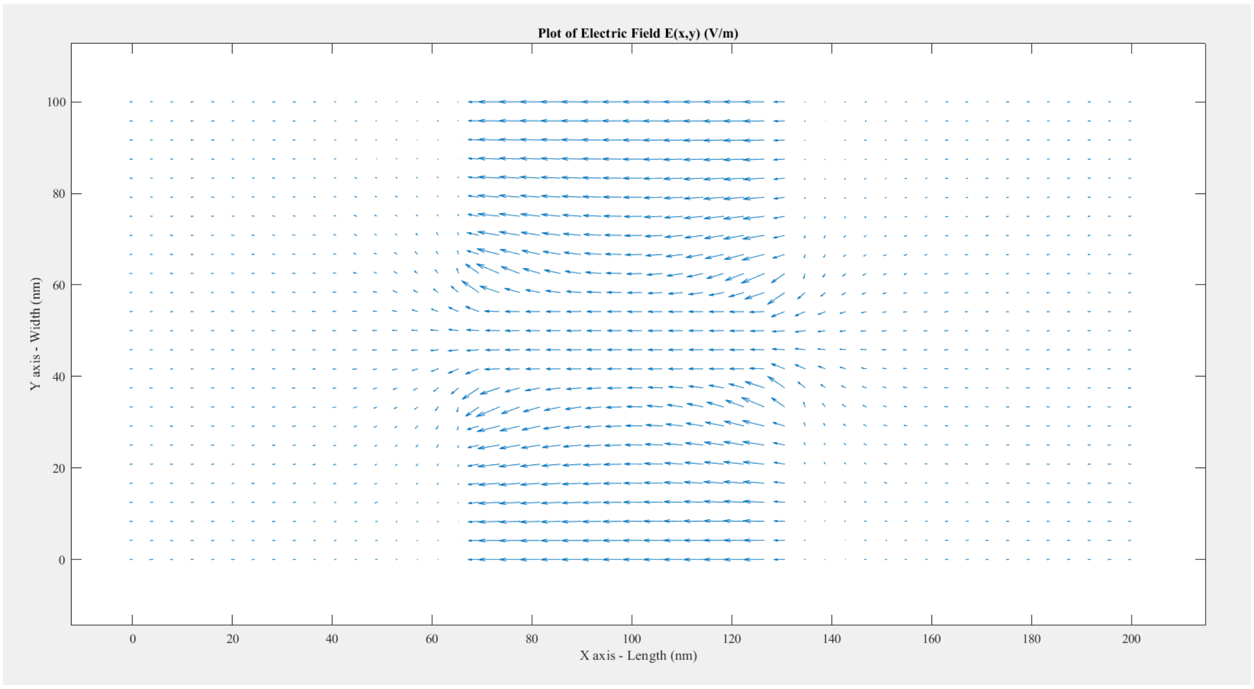

Figure 8 Vector plot for electric field

Q2 c)

Use the calculated field as input to the Monte-Carlo simulation.

```
global boxes;  % matrix for the boxes: n rows, and 4 columns for [x y w h]
% Initialize acceleration for each electron
ax = zeros(1, numE);  % Acceleration in x
ay = zeros(1, numE);  % Acceleration in y
% Calculate the acceleration field: a = Force/mass = q*E/mass
accFieldX = -C.q_0 * Ex / C.mn;
accFieldY = -C.q_0 * Ey / C.mn;

% Initialize the number of "super" electrons
numE = 10000;
numEPlot = 10;  % Number of electron to be plotted
% Number of simulation steps
numSim = 1000;
% Boudary mode: specular(0) or diffusive(1)
boundaryMode = 0;

% Add the boxes
numBox = AddObstacles(boxLF, boxWF, Region);

% Add the electrons
AddElectrons_WithBox(numE, Region, T, numBox);

% Calculate the scattering probability
Pscat = 1-exp(-deltaT/Tmn);

% Initalize plot
```

```
figure(10)
axCol = axes;
axCol.ColorOrder = rand(numEPlot,3); % Initalize color for each electron
hold on
% Draw the boxes
for iBox = 1:numBox
    rectangle("Position",boxes(iBox,:));
end

% Loop for simulation
for iSim = 1:numSim
     PlotPoint(numEPlot,numGridX, numGridY);

     % Store the current positions
     xp = x;
     yp = y;
     % Calculate the future positions: x = x0 + vx*t
     x = x + vx * deltaT;
     y = y + vy * deltaT;
     % Calculate the future velocity: vx = ax*t
     vx = vx + ax*deltaT;
     vy = vy + ay*deltaT;

    % Loop through all the particles
    for iE=1:numE
        % flag for invalid position
        bInvalid = false;

        % Step 1 - Check for boundary
        % Check for invalid x position
        if x(iE) <= 0
            x(iE) = Region.x; % Appear on right
            xp(iE) = x(iE);
            bInvalid = true;
        elseif x(iE) >= Region.x
            x(iE) = 0; % Appear on left
            xp(iE) = x(iE);
            bInvalid = true;
        end
        % Check for invalid y position
        if y(iE) <= 0
            bInvalid = true;
            y(iE) = 0;
            % Check for boundary mode
            if boundaryMode == 0  % Specular boundary
                vy(iE) = -vy(iE);
            else  % Diffusive boundary  TODO: check diffusive implementation
                vy(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vy
            end
        elseif y(iE) >= Region.y
            y(iE) = Region.y;
            bInvalid = true;
            % Check for boundary mode
            if boundaryMode == 0  % Specular boundary
```

```
                vy(iE) = -vy(iE);
            else  % Diffusive boundary
                vy(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vy
            end
        end

        % Step 2: Check for boxes
        for iBox = 1:numBox
            % Retrieve box info
            boxX1 = boxes(iBox, 1);
            boxX2 = boxes(iBox, 1)+boxes(iBox, 3);
            boxY1 = boxes(iBox, 2);
            boxY2 = boxes(iBox, 2)+boxes(iBox, 4);
            % Check if the particle is inside a box
            if (x(iE)>=boxX1 && x(iE)<=boxX2 && y(iE)>=boxY1 && y(iE) <= boxY2)
                bInvalid = true;   %Invalid position
                % Check for x position
                if xp(iE) <= boxX1  % Coming from left side
                    x(iE) = boxX1;
                    % Check for boundary mode
                    if boundaryMode == 0  % Specular boundary
                        vx(iE) = -vx(iE);
                    else  % Diffusive boundary
                        vx(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vx
                    end
                elseif xp(iE) >= boxX2  % Coming from right side
                    x(iE) = boxX2;
                    % Check for boundary mode
                    if boundaryMode == 0  % Specular boundary
                        vx(iE) = -vx(iE);
                    else  % Diffusive boundary
                        vx(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vx
                    end
                end
                % Check for y position
                if yp(iE) <= boxY1  % Coming from bottom
                    y(iE) = boxY1;
                    % Check for boundary mode
                    if boundaryMode == 0  % Specular boundary
                        vy(iE) = -vy(iE);
                    else  % Diffusive boundary
                        vy(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vy
                    end
                elseif yp(iE) >= boxY2 % Coming from top
                    y(iE) = boxY2;
                    % Check for boundary mode
                    if boundaryMode == 0  % Specular boundary
                        vy(iE) = -vy(iE);
                    else  % Diffusive boundary
                        vy(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vy
                    end
                end
                % Break the loop for box
                break;
            end
        end

        % Step 3: Check for scattering
        if ~bInvalid && Pscat > rand()
            % Rethermalize  TODO: Check rethermalize process is correct
            vx(iE) = sqrt(C.kb*T/C.mn).*randn();
            vy(iE) = sqrt(C.kb*T/C.mn).*randn();
        end

        % Step 4: Find acceleration
        % Find the corresponding index for the acceleration field
        indexX = ceil(x(iE)/(deltaXY*10^-9));
        indexY = ceil(y(iE)/(deltaXY*10^-9));
        % Check for invalid index
        if indexX <= 0
            indexX = 1;
        end
        if indexY <= 0
            indexY = 1;
        end
        % Assign the acceleration of the electron
        ax(iE) = accFieldX(indexX);
        ay(iE) = accFieldY(indexY);
    end

    % Pause some time
    pause(pauseTime);
end
```

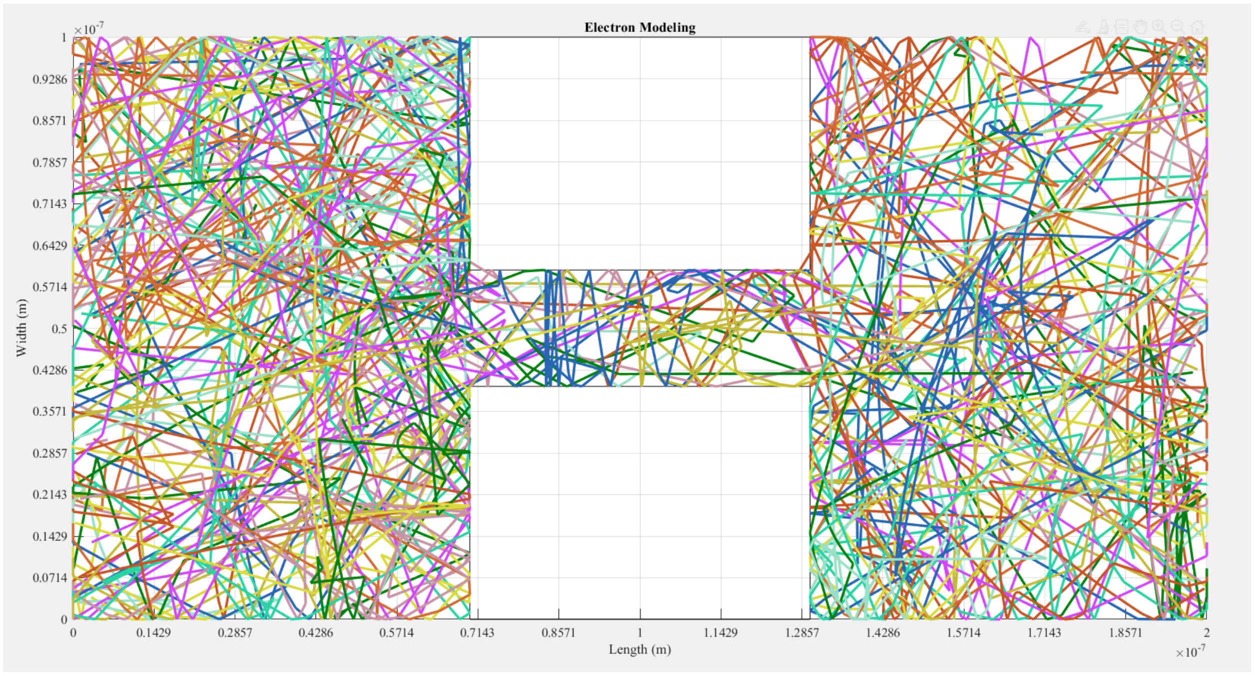

Figure 9 Plot of particle trajectories

Question 3

Use the coupled simulations to investigate the "device" and extract simple parameters

Q3 a)

In this part, the density map is plotted.

```
% Create the matrix for particle and total temperature
matrixParticles = zeros(numGridX+1,numGridY+1);

% Calculate the deltaX and deltaY for each grid
deltaX = Region.x/numGridX;
deltaY = Region.y/numGridY;

% Loop through all the electrons
for iE = 1:numE
    % Calculate the x index (column) in the tempeture matrix
    indexCol = floor(x(iE)/deltaX)+1;
    indexRow = floor(y(iE)/deltaY)+1;

    % Calculate the velocity squared
    Vsqrt = sqrt(vx(iE)^2 + vy(iE)^2);
    % Calculate the temperature
    T = C.mn * Vsqrt^2 / (2*C.kb);

    % Increment the particle matrix
    matrixParticles(indexRow, indexCol) = matrixParticles(indexRow, indexCol) + 1;
end

% Create the mesh grid
[X,Y] = meshgrid(linspace(0,Region.x,numGridX+1), linspace(0, Region.y, numGridY+1));

% Plot the density surface
figure(11)
surf(X,Y, matrixParticles);
view(0,90); % view from the top
title("Density Map")
xlabel("Length (m)")
ylabel("Width (m)")
snapnow
```

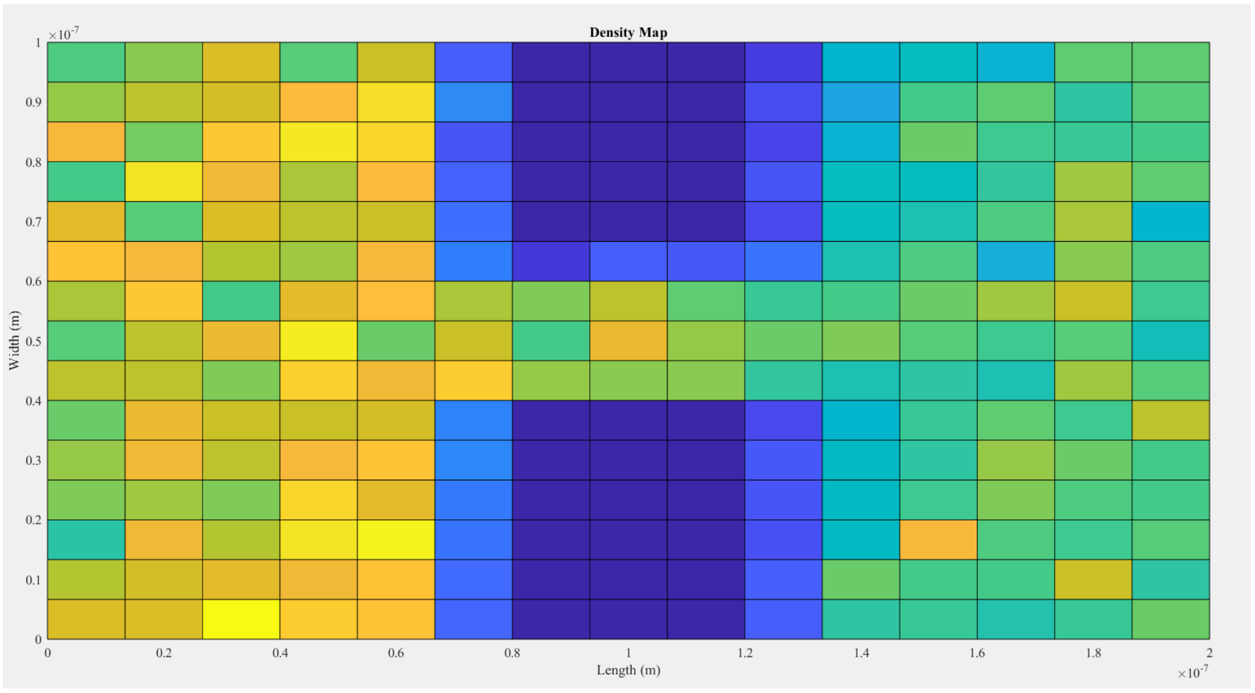

Figure 10 Plot of density map

Q3 a) Comment

With a high potential difference across the region, such as 0.8 V, the density on the left is higher than the density on the right. This is because the accelerations of the electrons are high, which resulting the speed of the electron is high. The electrons are bouncing off the obstacles with high speed, which lower the chance of the electron to pass the bottleneck. Therefore, the electron density is higher on one side than the other side.

Q3 b)

In this part, the average current at different bottleneck widths is calculated and plotted.

```
% Declare a vector for different box widths (in fraction)
vecBotNecWidths = linspace(0, 0.5, 10);
% Calculate the corresponding box width (in fraction)
vecBoxWidths = (ones(1,length(vecBotNecWidths)) - vecBotNecWidths) / 2;
% Vector to hold the current for different bottleneck width
vecBotNecCurrents = zeros(1, length(vecBotNecWidths));

% Loop through the different bottleneck widths
for ibotW = 1:length(vecBotNecWidths)
    % Step 1: Calculate the E field
    % Define the dimension
    L = Region.x * 10^9;  % Length in nm
    W = Region.y * 10^9;  % Width in nm
    boxLF = 0.3;  % Fraction of the length of the box
    boxWF = vecBoxWidths(ibotW);  % Fraction of the width of the box
    Lb = boxLF*L;  % Length of the box in nm
    Wb = boxWF*W;  % Width of the box in nm
```

```
deltaXY = 0.02*L;  % Assume deltaX = deltaY in nm
% Calculate the dimension of solution matrix
nx = (L/deltaXY);
ny = (W/deltaXY);
[X,Y] = meshgrid(linspace(0,L,nx), linspace(0,W,ny));
% Declare the matrix for conductivity: Sigma(y,x)
matrixSigma = ones(ny, nx);  % Dimension: ny times nx
xIndexBox = ceil((L-Lb)/(2*deltaXY));  % Find the starting x index for the box
LbIndexRange = ceil(Lb/deltaXY);  % Index range for the length of the box
WbIndexRange = ceil(Wb/deltaXY);  % Index range for the width of the box
% Assign the region for the box
matrixSigma(1:WbIndexRange, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;
matrixSigma(ny-WbIndexRange:ny, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;
% Declare the matrix for voltage V(y,x)
matrixV = zeros(ny, nx);  % Dimension: ny times nx
% Declare the G matrix and F vector: GV = F
G = zeros(nx*ny, nx*ny);
F = zeros(nx*ny, 1);
% Construct the G matrix and F vector
for ix = 1:nx
    for iy = 1:ny
        % Calculate the index
        n = mappingEq(ix, iy, ny);
        % Check for the boundary
        if ix==1 || ix==nx || iy ==1 || iy==ny
            G(n,n) = 1;
            % Boundary condition for x
            if ix == 1
                F(n,1) = voltageX0;  % V at x = 0
            elseif ix == nx
                F(n,1) = voltageX1;  % and V at x = L
            elseif iy == 1
                nyp = mappingEq(ix, iy+1, ny);  % dV/dy=0 at y=0
                G(n,nyp) = -1;
            elseif iy == ny
                nym = mappingEq(ix, iy-1, ny);  % dV/dy=0 at y=W
                G(n, nym) = -1;
            end
        else
            % Calculate the sigma
            sigmaxp = (matrixSigma(iy,ix) + matrixSigma(iy,ix+1))/2;
            sigmaxm = (matrixSigma(iy,ix) + matrixSigma(iy, ix-1))/2;
            sigmayp = (matrixSigma(iy,ix) + matrixSigma(iy+1, ix))/2;
            sigmaym = (matrixSigma(iy,ix) + matrixSigma(iy-1, ix))/2;
            % Calculate mapping index
            nxp = mappingEq(ix+1, iy, ny);  % index for V(i+1,j)
            nxm = mappingEq(ix-1, iy, ny);  % index for V(i-1,j)
            nyp = mappingEq(ix, iy+1, ny);  % index for V(i,j+1)
            nym = mappingEq(ix, iy-1, ny);  % index for V(i,j-1)
            % Setup the G matrix
            G(n,n) = -(sigmaxp+sigmaxm+sigmayp+sigmaym)/deltaXY^2;
            G(n, nxp) = sigmaxp/deltaXY^2;
            G(n, nxm) = sigmaxm/deltaXY^2;
            G(n, nyp) = sigmayp/deltaXY^2;
            G(n, nym) = sigmaym/deltaXY^2;
        end
    end
end
% Solve for V from GV = F
V = G\F;
% Map back to the 2D region
for iMap = 1:nx*ny
    % Calculate the index for the 2D region
    ix = ceil(iMap/ny);
    iy = mod(iMap, ny);
    if iy == 0
        iy = ny;
    end
    % Assign the value
    matrixV(iy, ix) = V(iMap);
end
% Solve the electric field
[Ex, Ey] = gradient(-matrixV);
Ex = Ex/(deltaXY * 10^-9);  % convert to V/m
Ey = Ey/(deltaXY * 10^-9);  % convert to V/m

% Step 2: Calculate the acceleration field
% Initialize the number of "super" electrons
numE = 1000;
numEPlot = 10;  % Number of electron to be plotted
% Number of simulation steps
numSim = 1000;
% Boudary mode: specular(0) or diffusive(1)
boundaryMode = 0;
% Add the boxes
numBox = AddObstacles(boxLF, boxWF, Region);
% To find the current, the following steps are performed:
% 1) Calculate the total area
areaA = Region.x * Region.y;  % m^2
areaA = areaA * 100^2;  % cm^2
% 2) Calculate the total electrons in the area assuming electron
% concentration is 10^15 cm-2
totalE = 10^15 * areaA;  % total electrons
% 3) Find the charge per "Super Electron", where "Super Electron" is the
% particle in this simulation
superECharge = -C.q_0 * totalE/numE;  % Charge per super electron
% 4) The current can be found by counting the net number of super electrons

% Initialize acceleration for each electron
ax = zeros(1, numE);  % Acceleration in x
ay = zeros(1, numE);  % Acceleration in y
% Calculate the acceleration field: a = Force/mass = q*E/mass
accFieldX = -C.q_0 * Ex / C.mn;
accFieldY = -C.q_0 * Ey / C.mn;

% Add the electrons
AddElectrons_WithBox(numE, Region, T, numBox);
% Calculate the scattering probability
```

```
Pscat = 1-exp(-deltaT/Tmn);

% Super electron count for current calculation
% Count on left side x=0. +1 flow right, -1 flow left
countECurrent = 0;  % Hold the super electron count

% Step 3: Loop for simulation
for iSim = 1:numSim
     % Store the current positions
     xp = x;
     yp = y;
     % Calculate the future positions: x = x0 + vx*t
     x = x + vx * deltaT;
     y = y + vy * deltaT;
     % Calculate the future velocity: vx = ax*t
     vx = vx + ax*deltaT;
     vy = vy + ay*deltaT;
    % Loop through all the particles
    for iE=1:numE
        % flag for invalid position
        bInvalid = false;
        % Step 1 - Check for boundary
        % Check for invalid x position
        if x(iE) <= 0
            x(iE) = Region.x; % Appear on right
            xp(iE) = x(iE);
            bInvalid = true;
            % Update the electron count for current calculation
            countECurrent = countECurrent-1;  % -1 flow left
        elseif x(iE) >= Region.x
            x(iE) = 0; % Appear on left
            xp(iE) = x(iE);
            bInvalid = true;
            % Update the electron count for current calculation
            countECurrent = countECurrent+1;  % +1 flow right
        end
        % Check for invalid y position
        if y(iE) <= 0
            bInvalid = true;
            y(iE) = 0;
            % Check for boundary mode
            if boundaryMode == 0  % Specular boundary
                vy(iE) = -vy(iE);
            else  % Diffusive boundary  TODO: check diffusive implementation
                vy(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vy
            end
        elseif y(iE) >= Region.y
            y(iE) = Region.y;
            bInvalid = true;
            % Check for boundary mode
            if boundaryMode == 0  % Specular boundary
                vy(iE) = -vy(iE);
            else  % Diffusive boundary
                vy(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vy
        end
    end
    % Step 2: Check for boxes
    for iBox = 1:numBox
        % Retrieve box info
        boxX1 = boxes(iBox, 1);
        boxX2 = boxes(iBox, 1)+boxes(iBox, 3);
        boxY1 = boxes(iBox, 2);
        boxY2 = boxes(iBox, 2)+boxes(iBox, 4);
        % Check if the particle is inside a box
        if (x(iE)>=boxX1 && x(iE)<=boxX2 && y(iE)>=boxY1 && y(iE) <= boxY2)
            bInvalid = true;   %Invalid position
            % Check for x position
            if xp(iE) <= boxX1  % Coming from left side
                x(iE) = boxX1;
                % Check for boundary mode
                if boundaryMode == 0  % Specular boundary
                    vx(iE) = -vx(iE);
                else  % Diffusive boundary
                    vx(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vx
                end
            elseif xp(iE) >= boxX2  % Coming from right side
                x(iE) = boxX2;
                % Check for boundary mode
                if boundaryMode == 0  % Specular boundary
                    vx(iE) = -vx(iE);
                else  % Diffusive boundary
                    vx(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vx
                end
            end
            % Check for y position
            if yp(iE) <= boxY1  % Coming from bottom
                y(iE) = boxY1;
                % Check for boundary mode
                if boundaryMode == 0  % Specular boundary
                    vy(iE) = -vy(iE);
                else  % Diffusive boundary
                    vy(iE) = -abs(sqrt(C.kb*T/C.mn).*randn());  % negative vy
                end
            elseif yp(iE) >= boxY2 % Coming from top
                y(iE) = boxY2;
                % Check for boundary mode
                if boundaryMode == 0  % Specular boundary
                    vy(iE) = -vy(iE);
                else  % Diffusive boundary
                    vy(iE) = abs(sqrt(C.kb*T/C.mn).*randn());  % positive vy
                end
            end
            % Break the loop for box
            break;
        end
    end
    % Step 3: Check for scattering
    if ~bInvalid && Pscat > rand()
```

```
                % Rethermalize  TODO: Check rethermalize process is correct
                vx(iE) = sqrt(C.kb*T/C.mn).*randn();
                vy(iE) = sqrt(C.kb*T/C.mn).*randn();
            end
            % Step 4: Find acceleration
            % Find the corresponding index for the acceleration field
            indexX = ceil(x(iE)/(deltaXY*10^-9));
            indexY = ceil(y(iE)/(deltaXY*10^-9));
            % Check for invalid index
            if indexX <= 0
                indexX = 1;
            end
            if indexY <= 0
                indexY = 1;
            end
            % Assign the acceleration of the electron
            ax(iE) = accFieldX(indexX);
            ay(iE) = accFieldY(indexY);
        end
    end

    % Calculate the current
    vecBotNecCurrents(ibotW) = superECharge*countECurrent/deltaT;

end

% Plot the current vs bottleneck
figure(12)
% convert bottle neck widths from fraction to nm
vecBotNecWidths = vecBotNecWidths * Region.x * 10^9;
plot(vecBotNecWidths, abs(vecBotNecCurrents), "-b.")
title("Current vs bottleneck width")
xlabel("bottleneck width (nm)")
ylabel("Current Magnitude (A)")
grid on
```

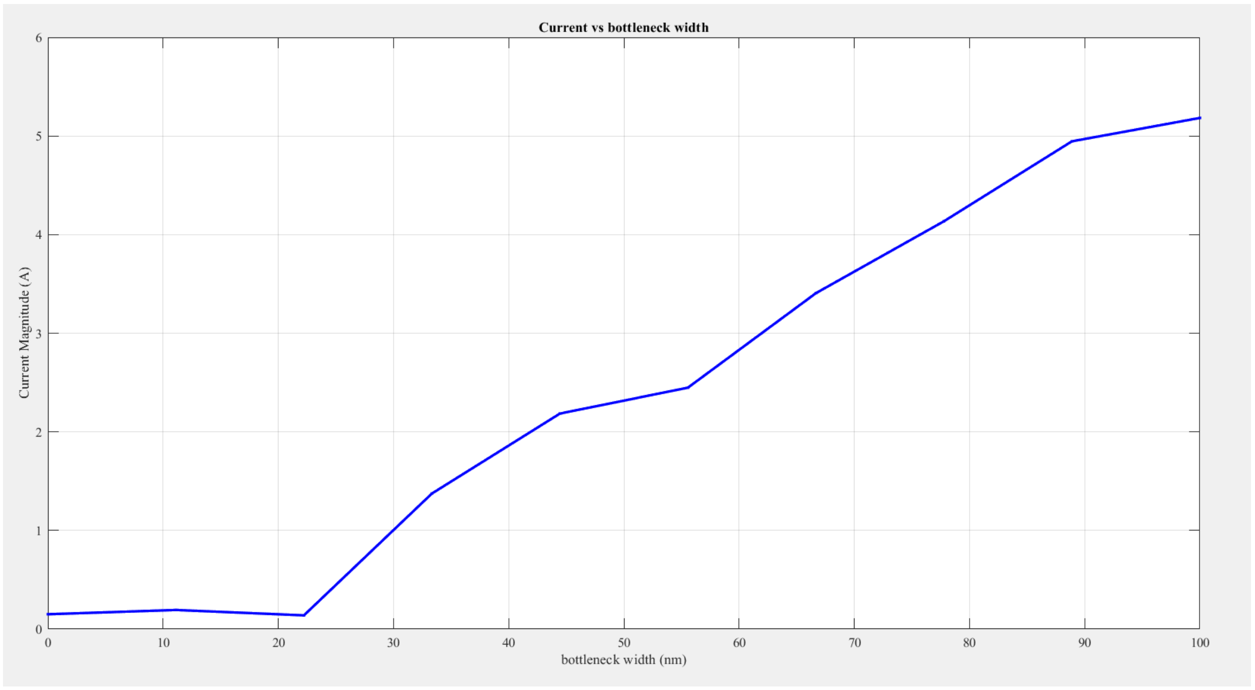

Figure 11 Plot of current versus bottleneck

Q3 b) comment

The plot of current vs bottleneck width shows that the current is increasing as the bottleneck width is increased. This make sense since a larger opening will allow more electrons to flow through the bottle neck more easily, so the current is expected to increase as the bottle neck increased. Please note that the current is measured at the instance of the end of the simulation for each bottleneck width, so the trend of the current plot contains some irregularities due to “noises”. Nevertheless, the plot of current versus bottleneck do shows that the current is increasing as the bottleneck with is increased.

Q3 c)

The next step to make this simulation more accurate can be to add the interactions between the electrons. The simulation now ignored the interactions between the electrons. Since electrons have the same negative charges, it is expected that they will repel each other. By adding the interactions between the electrons can make this simulation more accurate and closer to reality.

Helper functions

The following functions are the helper functions used in the main code.

```
function AddElectrons(numE, region, vth, T)
global C  % Constants
global x y % arrays for current electron positions
global xp yp % arrays for previous electron positions
global vx vy % arrays for current electron  velocities

% Create the arrays for electrons locations
x = rand(1, numE) * region.x;
xp = x;
```

```matlab
y = rand(1, numE) * region.y;
yp = y;

% TODO: Verify that this is Maxwell-Boltzmann distribution
% mean of vth and standard deviation of sqrt(kT/m)
% Initialize the arrays for velocity distrubution
vx = sqrt(C.kb*T/C.mn).*randn(1, numE);
vy = sqrt(C.kb*T/C.mn).*randn(1, numE);

% Plot the Vth distribution
plotVthDistribution(30);

% Display the vth to compare with the actual distribution
fprintf("vth = "+vth + " m/s\n");
end % End AddElectrons


% Helper function to plot one point for electron position
% @param  numEPlot = number of electrons to be plotted
%         numGridX = number of grid on the x axis
%         numGridY = number of grid on the y axis
function PlotPoint(numEPlot, numGridX, numGridY)
global x y xp yp limits

% plot the electron positions
plot([xp(1:numEPlot);x(1:numEPlot)], [yp(1:numEPlot);y(1:numEPlot)])

% Adjust the axis limits
axis(limits)
% Set grid
set(gca,'xtick',linspace(0, limits(2), numGridX));
set(gca, 'ytick',linspace(0, limits(4), numGridY));
grid on

% Add title and labels
title("Electron Modeling");
xlabel("Length (m)")
ylabel("Width (m)")
end  % End PlotPoint


% Helper function to plot the vth distribution
% @ param  nbins = number of bins
function plotVthDistribution(nbins)
global vx vy

% Calculate the vth
Vth_data = sqrt(vx.^2 + vy.^2);

% Plot the velocity distribution histogram
figure(1)
hist(Vth_data, nbins);
title("Vth Distribution")
ylabel("Counts (number)")
```

```
xlabel("Velocity (m/s)")
snapnow
end  % End plotVthDistribution


% This function generate a 2D temperature color plot
% @param numGridX = number of grid in the x direction
%        numGridY = number of grid in the y direction
%        numE = number of electrons
%        limitX = region limit on the x axis
%        limitY = region limit on the y axis
function tempDisplay(numGridX, numGridY, numE, limitX, limitY)
% Global varibles use for temperature calculation
global x y vx vy C
global limits

% Create the matrix for particle and total temperature
matrixParticles = zeros(numGridX+1,numGridY+1);
matrixTempTotal = zeros(numGridX+1, numGridY+1);

% Calculate the deltaX and deltaY for each grid
deltaX = limitX/numGridX;
deltaY = limitY/numGridY;

% Loop through all the electrons
for iE = 1:numE
    % Calculate the x index (column) in the tempeture matrix
    indexCol = floor(x(iE)/deltaX)+1;
    indexRow = floor(y(iE)/deltaY)+1;

    % Calculate the velocity squared
    Vsqrt = sqrt(vx(iE)^2 + vy(iE)^2);
    % Calculate the temperature
    T = C.mn * Vsqrt^2 / (2*C.kb);

    % Increment the total temperature matrix
    matrixTempTotal(indexRow, indexCol) = matrixTempTotal(indexRow, indexCol) + T;
    % Increment the particle matrix
    matrixParticles(indexRow, indexCol) = matrixParticles(indexRow, indexCol) + 1;
end

% Create the mesh grid
[X,Y] = meshgrid(linspace(0,limitX,numGridX+1), linspace(0, limitY, numGridY+1));

% Plot the density surface
figure(4)
surf(X,Y, matrixParticles);
view(0,90); % view from the top
title("Density Map")
xlabel("Length (m)")
ylabel("Width (m)")
snapnow

% Calculate the temperature matrix
```

```matlab
Temp = matrixTempTotal ./ matrixParticles;
Temp(isnan(Temp)) = 0;

% Plot the temperature surface
figure(5)
surf(X,Y,Temp);
view(0,90); % view from the top
title("Temperature Map")
xlabel("Length (m)")
ylabel("Width (m)")
snapnow
end  % End tempDisplay


% Helper function for mapping index
% @param iRow = i index for the row
%        jRow = j index for the column
%        ny   = size of the y
function [n] = mappingEq(iRow, jCol, ny)
    n = jCol + (iRow - 1) * ny;
end  % End mappingEq


% Helper function to add the obstacles
% @ param  boxLF = length of the box in fraction of region.x
%          boxWF = width of the box in fraction of region.y
%          region = region.x and region.y
function [numBox] = AddObstacles(boxLF, boxWF, region)
global boxes  % Matrix for holding the boxes
% Find the x, y, w, h for the bottom box
xbb = region.x/2 - region.x*boxLF/2;
ybb = 0;
wbb = region.x*boxLF;
hbb = region.y * boxWF;
% Find the x, y, w, h for the upper box
xub = region.x/2 - region.x * boxLF/2;
yub = region.y * (1-boxWF);
wub = region.x * boxLF;
hub = region.y * boxWF;

% Create the boxes
boxes = [xbb ybb wbb hbb;
         xub yub wub hub];
% Return number of boxes
numBox = height(boxes);
end  % End AddObstacles

% This function add a bunch of electrons in a given region randomly for Q3
% @param numE = number of electrons
%        region = region for the electrons
%        T = temperature in Kelvin
%        numBox = number of boxes
function AddElectrons_WithBox(numE, region, T, numBox)
global C  % Constants
global x y % arrays for current electron positions
global xp yp % arrays for previous electron positions
global vx vy % arrays for current electron velocities
global boxes  % Matrix for the boxes position

% Create the arrays for electrons locations
x = rand(1, numE) * region.x;
y = rand(1, numE) * region.y;

% Loop through the electrons to make sure that no electrons inside obstacles
for iE = 1:numE
    % Flag to indicate whether inside box
    insideBox = true;
    while (insideBox)
        insideBox = false;
        % Loop through the boxes
        for iBox = 1:numBox
            % Check for invalid electrons position
            if (x(iE)>boxes(iBox, 1) && x(iE)<(boxes(iBox, 1)+boxes(iBox, 3)) ...
                    && y(iE)>boxes(iBox, 2) && y(iE) < (boxes(iBox, 2)+boxes(iBox, 4)))
                insideBox = true;
                break;
            end
        end
        if (insideBox)
            % Regenerate position
            x(iE) = rand() * region.x;
            y(iE) = rand() * region.y;
        end
    end
end
% Create the arrays for previous electron positions
xp = x;
yp = y;
% Create helper arrays for velocity distrubution
vx = sqrt(C.kb*T/C.mn).*randn(1, numE);
vy = sqrt(C.kb*T/C.mn).*randn(1, numE);
end  % End AddElectrons_WithBox
```

Published with MATLAB® R2021b