Table of Contents

ELEC4700 Assignment 1: Monte-Carlo Modeling of Electron Transport

Dilsha Appu-Hennadi 101107857

This assignment will be walking through various exercises to model carriers as a population of electrons in an N-type Si semiconductor crystal. For the purposes of this assignment, electron mass is defined as m_n = 0.26*m_0, and the region will be 200nm X 100nm.

```
clear all
clearvars
clearvars -GLOBAL
close all
format shorte

% set(0, 'DefaultFigureWindowStyle', 'docked')

global m_n nomRegionL nomRegionW C B
global x y Vx Vy nAtoms numElec nPlottedElec numTimeStep dt

C.q_0 = 1.60217653e-19; % Electron charge
C.hb = 1.054571596e-34; % Dirac constant
C.h = C.hb * 2 * pi;    % Planck constant
C.m_0 = 9.10938215e-31; % Rest mass of an electron
C.kb = 1.3806504e-23;   % Boltzmann constant
C.eps_0 = 8.854187817e-12; % Vacuum permittivity
C.mu_0 = 1.2566370614e-6; % Vacuum permeability
C.c = 299792458;        % Speed of light
C.g = 9.80665;          % Acceleration due to gravity (m/s^2)
C.am = 1.66053892e-27;

m_n = 0.26*C.m_0;         % Effective mass of electrons
nomRegionL = 200e-9;       % Nominal region length (nm)
nomRegionW = 100e-9;       % Nominal region width (nm)

numElec = 100;
nPlottedElec = 0.3*numElec;
numTimeStep = 1600;
dt = 1e-15;

% newcolours = ['r','g','b','c','m','y','k'];
```

1 ELECTRON MODELING

1. Thermal velocity, assuming a temperature of T = 300K

```
Temp = 300;
v_th = sqrt(C.kb * Temp / m_n);
```

Given a temperature of 300K, the thermal velocity is calculated to be about 1.3224e5 m/s

2. If the mean time between collisions is t_mn = 0.2 ps, what is the mean free path?

```
t_mn = 0.2e-12;
L_n = v_th*t_mn; %m
```

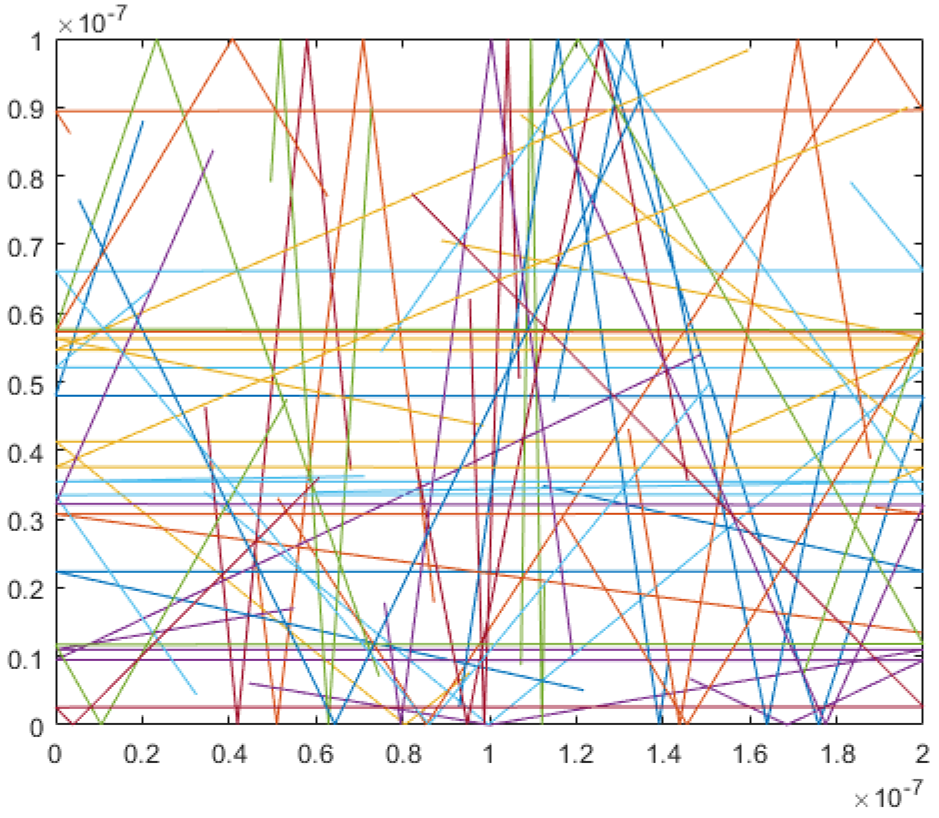
The mean free path is taken as the thermal velosity times the mean time between collisions, which gives us a distance of 2.6449e-8 m.

3. Write a program to model the random motion of electrons.

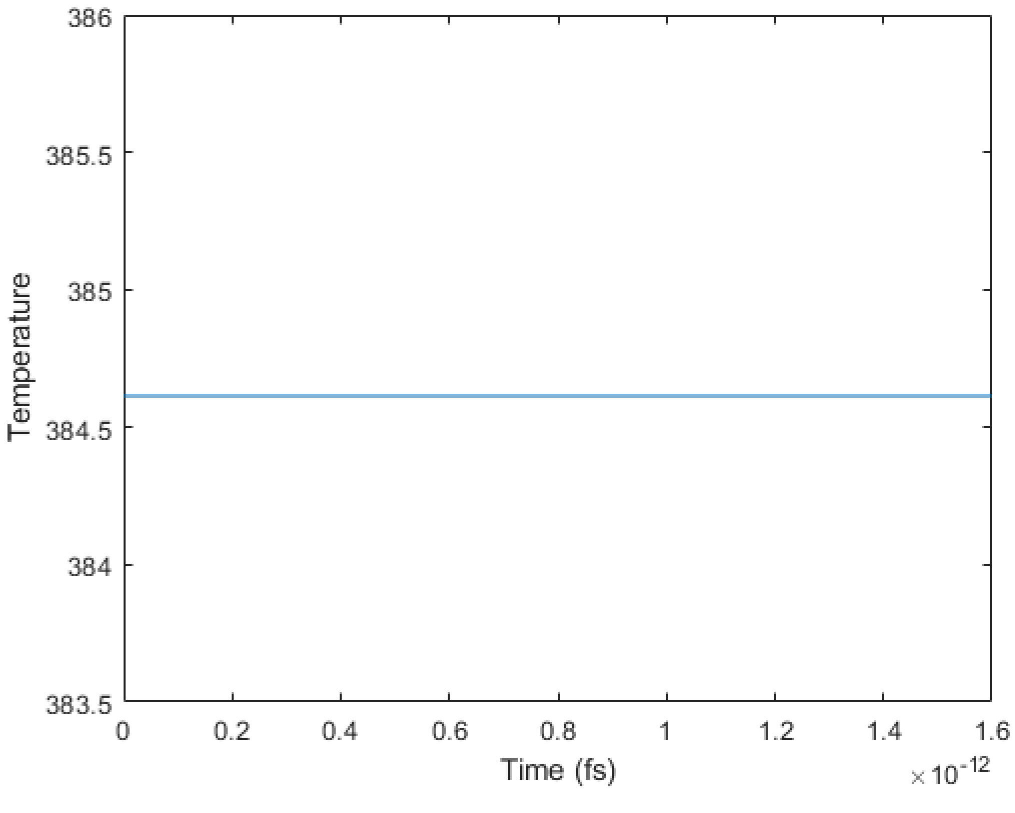
```
currX = (nomRegionL).*rand(numElec, 1); % set random initial x
 position
currY = (nomRegionW).*rand(numElec, 1); % set random initial y
 position
currVel = zeros(numElec,1); % initialize column vector to hold
 velocity, velocities initialized with zero
currVel(:,1) = v_th; % set all values in the first column as v_th
currDir = (2*pi).*rand(numElec,1); % create column vector with current
 direction of each electron
currVX = []; % set up x and y velocity vectors as empity column vector
currVY = [];

% Calculate initial vx and vy for each electron given their direction
[currVX, currVY] = XYVelocities(currVel,currDir,numElec);
```

At this point we should have the initial x and y locations of all electron. The locations should be within the bounded region. All electrons should have their initial velocity set to the thermal velocity, v_th and a random direction between 0 to 2*pi. From the magnitude of the velocity and the direction, the x and y vector components of velocity can be determined. From here, we can now see how the electrons will move about the region. To plot trajectories, we need to save the previous locations of each electrons - i.e need a matrix where rows are individual electrons and columns are different times - these will be saved in a separate matric from the currXXX column vectors.

```
for n = 0:numTimeStep

    currTime(n+1) = n*dt; % determine current time (fs)

    % Update according to Newton's laws of motion
    % new position = old position + velocity*time

    if n > 0 %update position after t=0

        currX(:,1) = currX(:,1) + currVX(:,1)*dt; % calculate new X
        % if e- crosses right bound, set X=0; if e- crosses left
 bound, set
        % X=upper bound
        crossRight = currX > nomRegionL;
        currX(crossRight) = 0;
        crossLeft = currX < 0;
        currX(crossLeft) = nomRegionL;

        newY = currY + currVY*dt; % check if new Y crosses boundary
        bounce = (newY>nomRegionW) | (newY<0); % bounce electron if it
 hits bounds
        currVY(bounce) = -currVY(bounce);
        currY(:,1) = currY(:,1) + currVY(:,1)*dt;

    end

    % save previous position to plot trajectory
    prevX(:,n+1) = currX;
    prevY(:,n+1) = currY;

    % Calculate average kinetic energy of electrons:
    avgE_k = C.m_0*(sum(sqrt(currVX.^2 + currVY.^2).^2)/numElec)/2;
    % Calculate temperature of material:
    currTemp(n+1) = (2*avgE_k)/(3*C.kb);

end

% make plot

figure(1)
% Colour = hsv(nPlottedElec);
for i = 1:nPlottedElec

    plot(prevX(i,:),prevY(i,:))
    hold on

end
hold off
axis([0 200e-9 0 100e-9])

figure(2)
plot(currTime,currTemp)
xlabel('Time (fs)')
ylabel('Temperature')
```



×10^-7
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^-7



386
385.5
385
384.5
384
383.5
Temperature
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time (fs)
×10^-12

2 Collisions with Mean Free Path (MFP)

We reset the position and velocity variables from the previous section for use here.

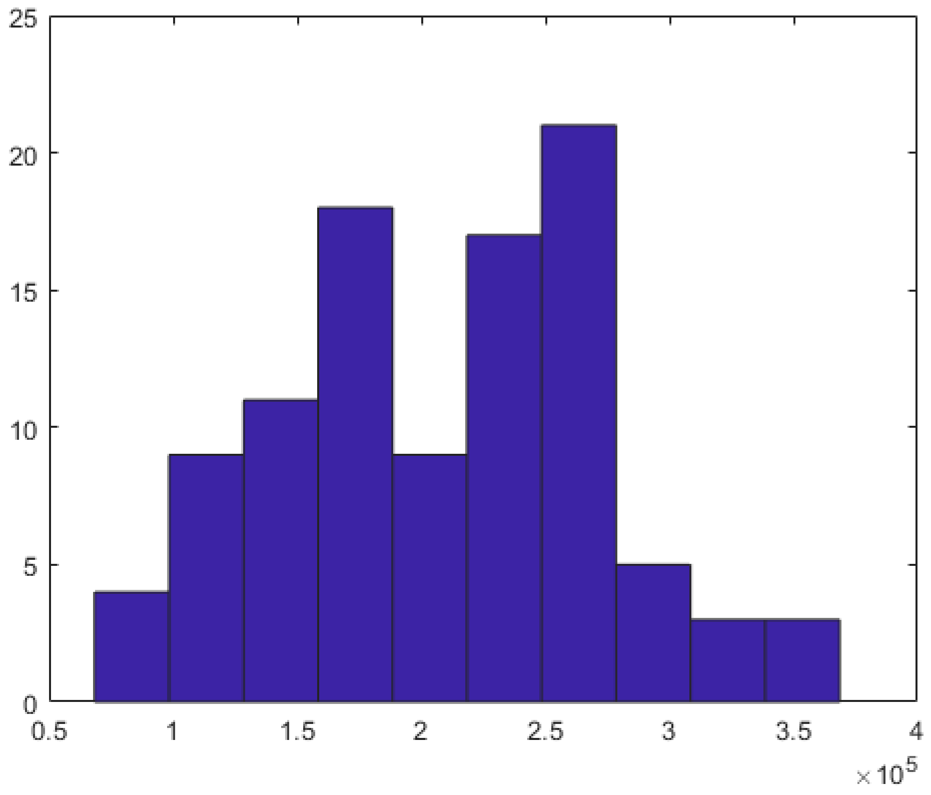
```
currX = (nomRegionL).*rand(numElec, 1); % set random initial x
 position
currY = (nomRegionW).*rand(numElec, 1); % set random initial y
 position
currVel = zeros(numElec,1); % initialize column vector to hold
 velocity, velocities initialized with zero
currDir = (2*pi).*rand(numElec,1); % create column vector with current
 direction of each electron
currVX = []; % set up x and y velocity vectors as empity column vector
currVY = [];
prevX = []; % clear vars
prevY = [];
currTime = [];
randVal = [];
bounce = [];
numScat = [];
```

To calculate velocity vectors, the vector components are randomly assigned according to the Boltzmann-Maxwell Distribution. For velocity components, this distribution is Gaussian with a standard deviation of sqrt(k*T/m). The distribution is also adjusted such that the average velocity is the thermal velocity.

```
currVX = v_th + sqrt(C.kb*Temp/C.m_0)*randn(numElec,1);
currVY = v_th + sqrt(C.kb*Temp/C.m_0)*randn(numElec,1);
```

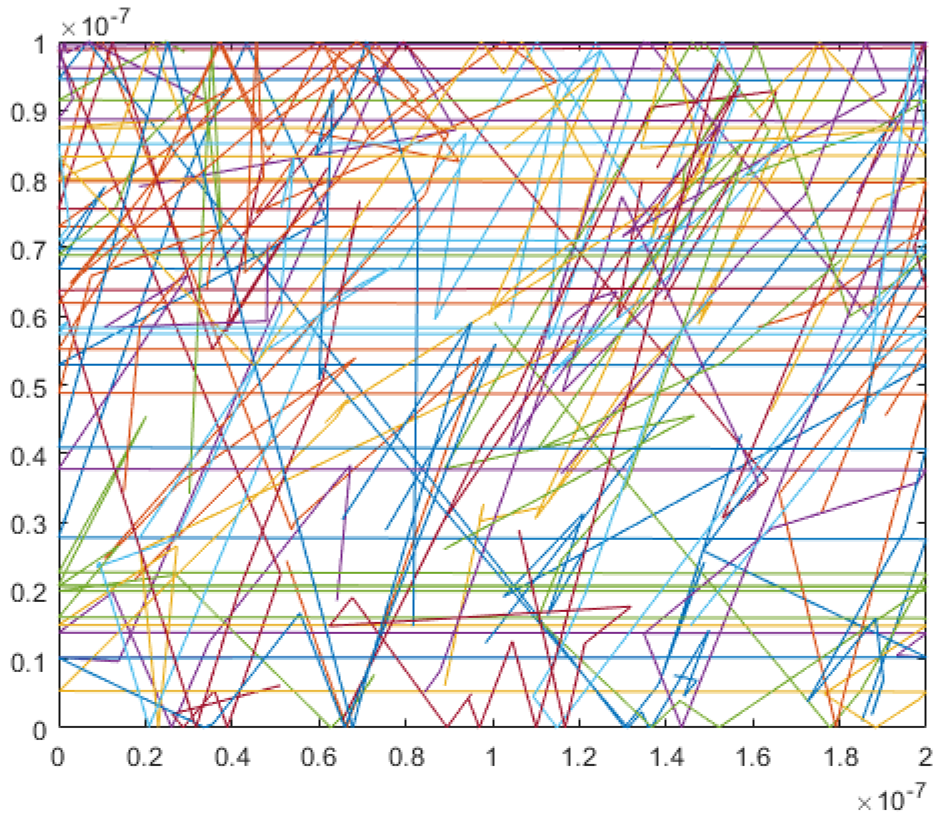
We check the distribution of the velocities by obtaining the magnitude of the velocities and plotting them into a histogram.

```
currVel = sqrt(currVX.^2 + currVY.^2);

figure(3)
hist(currVel);
```



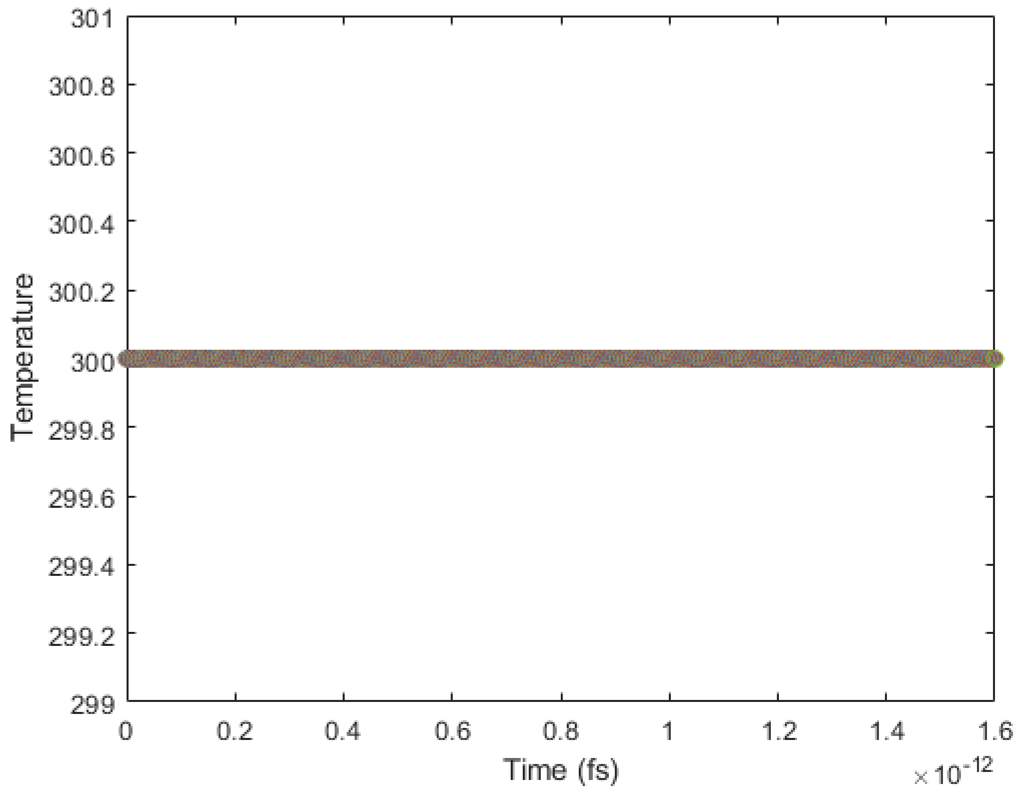
In this section, we will be studying the effects of scattering. The probability of scattering is given by P_scat = 1 - exp(-dt/t_mn). We can use this formula to obtain the probability of scattering within our system. We can also obtain a rough idea of the scattering of electrons by observing a random electron. In this case, we will observe the scattering of electron 1.

```
P_scat = 1 - exp(-(dt)/t_mn);
numScat = zeros(1,numTimeStep+1); % just look at scattering of
 electron 1

for n = 0:numTimeStep

    currTime(n+1) = n*dt; % determine current time (ms)

    % Update according to Newton's laws of motion
    % new position = old position + velocity*time

    if n > 0 %update position after t=0

        randVal = rand(numElec,1); % assign scatter probability
        scatter = randVal<=P_scat;
        currVX(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;
        currVY(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;

        % chance to invert direction when scattering
        randVal = rand(numElec,1);
        invertDir = scatter & (randVal<=0.5);
        currVX(invertDir) = -currVX(invertDir);
        currVY(invertDir) = -currVY(invertDir);

        % if electron 1 scatters, count it
        if scatter(1,1)
            numScat(n+1) = numScat(n+1) + 1;
        end

        currX(:,1) = currX(:,1) + currVX(:,1)*dt; % calculate new X
        % if e- crosses right bound, set X=0; if e- crosses left
 bound, set
        % X=upper bound
        crossRight = currX > nomRegionL;
        currX(crossRight) = 0;
        crossLeft = currX < 0;
        currX(crossLeft) = nomRegionL;

        newY = currY + currVY*dt; % check if new Y crosses boundary
        bounce = (newY>nomRegionW) | (newY<0); % bounce electron if it
 hits bounds
        currVY(bounce) = -currVY(bounce);
        currY(:,1) = currY(:,1) + currVY(:,1)*dt;

    end

    % save previous position to plot trajectory
    prevX(:,n+1) = currX;
    prevY(:,n+1) = currY;

    currVel = sqrt(currVX.^2 + currVY.^2);

    % Calculate average kinetic energy of electrons:
    avgE_k = C.m_0*(sum(currVel.^2)/numElec)/2;
    % Calculate temperature of material:
    currTemp(n+1) = (2*avgE_k)/(3*C.kb);

end

% make plot

figure(4)
for i = 1:nPlottedElec
    plot(prevX(i,:),prevY(i,:))
    hold on
end
hold off
```



Throughout the iterations, the temperature of the system was taken. With the temperature at each time step recorded, the data can be plotted to observe the temperature of the system over time.

```
figure(5)
plot(currTime,Temp,'o-')
xlabel('Time (fs)')
ylabel('Temperature')
```



To calculate the mean time between collisions, we identify the times a collision occurs for an electron. Then, we can look at the difference in time between scattering events and take the average. Then obtain the mean free path as the product of the electron's velocity and mean time between collisions.

```
deltaT = 0; % initialized time
sumT = 0;

for t = 0:(numTimeStep-1)

    if (numScat(t+1) > 0)

        sumT = sumT + deltaT; % every time a scattering event occured
 take record of how much time had passed
        deltaT = 0; % reset "timer"

    end

    deltaT = deltaT + dt;

end
```

calculate mean time between collisions by dividing the sum of time between collisions by the number of collision events.

```
calc_t_mn = sumT/sum(numScat);

calc_L_n = v_th*calc_t_mn;
```

3 Enhancements

In this section, we will add boxes to the system which electrons will not be able to pass through. We can define the sides of the boxes as occuring at some fraction of the region length/width.

```
% Define boxes
% Box 1 (top)
B.Left1 = 0.35;
B.Right1 = 0.65;
B.Top1 = 1;
B.Bottom1 = 0.6;

% Box 2 (bottom)
B.Left2 = 0.35;
B.Right2 = 0.65;
B.Top2 = 0.5;
B.Bottom2 = 0;

currX = (nomRegionL).*rand(numElec, 1); % set random initial x
 position
currY = (nomRegionW).*rand(numElec, 1); % set random initial y
 position
```

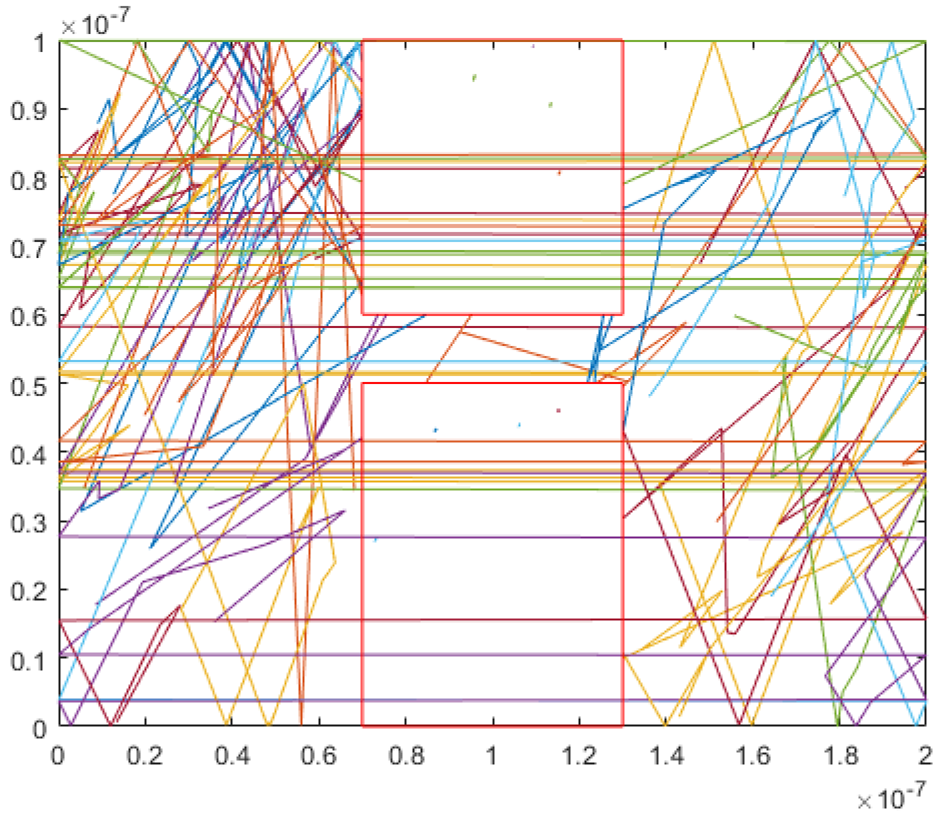
Before moving forward, we must ensure all initial positions lie outisde the boxes.

```
for i = 1:numElec
    while ((currX(i)>=(B.Left1*nomRegionL))
 && (currX(i)<=(B.Right1*nomRegionL)) &&
 ((currY(i)>=(B.Bottom1*nomRegionW)) ||
 (currY(i)<=(B.Top2*nomRegionW))))
        currX = (nomRegionL).*rand(numElec, 1); % set random initial x
 position
        currY = (nomRegionW).*rand(numElec, 1); % set random initial y
 position
    end
end

currVel = zeros(numElec,1); % initialize column vector to hold
 velocity, velocities initialized with zero
currDir = (2*pi).*rand(numElec,1); % create column vector with current
 direction of each electron
currVX = []; % set up x and y velocity vectors as empity column vector
currVY = [];
prevX = []; % clear vars
prevY = [];
currTime = [];
randVal = [];
bounce = [];

% Calculate velocity vectors
currVX = v_th + sqrt(C.kb*Temp/C.m_0)*randn(numElec,1);
currVY = v_th + sqrt(C.kb*Temp/C.m_0)*randn(numElec,1);
```

The probability of scattering is given as: P_scat = 1 - exp(-dt/t_mn)

```
P_scat = 1 - exp(-(dt)/t_mn);

for n = 0:numTimeStep

    currTime(n+1) = n*dt; % determine current time (ms)

    % Update according to Newton's laws of motion
    % new position = old position + velocity*time

    if n > 0 %update position after t=0

        randVal = rand(numElec,1); % assign scatter probability
        scatter = randVal<=P_scat;
        currVX(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;
        currVY(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;

        % chance to invert direction when scattering
        randVal = rand(numElec,1);
        invertDir = scatter & (randVal<=0.5);
        currVX(invertDir) = -currVX(invertDir);
        currVY(invertDir) = -currVY(invertDir);

        checkBounce;

        % if e- crosses right bound, set X=0; if e- crosses left
 bound, set
        % X=upper bound
        crossRight = newX > nomRegionL;
        currX(crossRight) = 0;
        crossLeft = newX < 0;
        currX(crossLeft) = nomRegionL;

    end

    % save previous position to plot trajectory
    prevX(:,n+1) = currX;
    prevY(:,n+1) = currY;

    currVel = sqrt(currVX.^2 + currVY.^2);

    % Calculate average kinetic energy of electrons:
    avgE_k = C.m_0*(sum(currVel.^2)/numElec)/2;
    % Calculate temperature of material:
    currTemp(n+1) = (2*avgE_k)/(3*C.kb);

end

% make plot

figure(6)
for i = 1:nPlottedElec
    plot(prevX(i,:),prevY(i,:))
    hold on
end
```

```
makeBox(B.Left1,B.Right1,B.Top1,B.Bottom1,nomRegionL,nomRegionW);
makeBox(B.Left2,B.Right2,B.Top2,B.Bottom2,nomRegionL,nomRegionW);
hold off
```



3.1 Injection Model

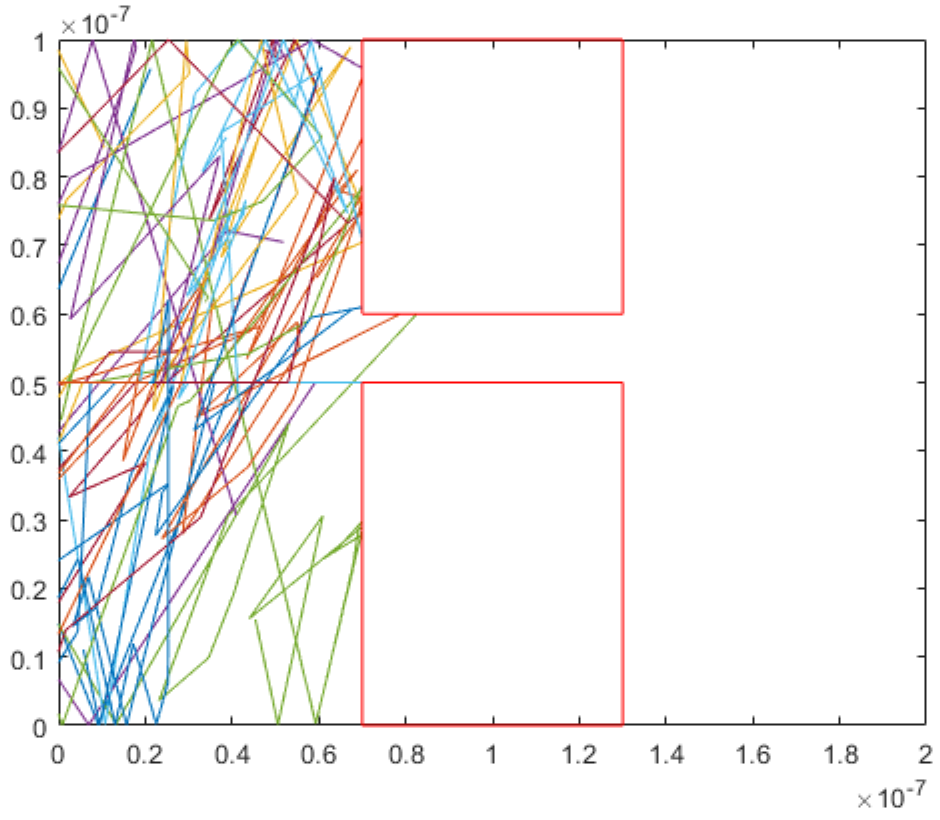
Continuing on from the boxes, we'll now simulate an injection of electrons into the system. The electrons will start out on the left side of the region at the mid-way point of the width and will be sent in with a positive x velocity and 0 y velocity component. Right away the electrons may experience scattering.

```
currX = zeros(numElec,1); % start electrons a left side of the screen
currY = 0.5*nomRegionW.*ones(numElec, 1); % middle of the region width
currVel = zeros(numElec,1); % initialize column vector to hold
 velocity, velocities initialized with zero
currDir = (2*pi).*rand(numElec,1); % create column vector with current
 direction of each electron
currVX = []; % set up x and y velocity vectors as empity column vector
currVY = [];
prevX = []; % clear vars
prevY = [];
currTime = [];
randVal = [];
bounce = [];

% Calculate velocity vectors
currVX = v_th + sqrt(C.kb*Temp/C.m_0)*randn(numElec,1);
```

```
currVY = zeros(numElec,1); % no Y component to velocity vector
 initially
```

The probability of scattering is given as: P_scat = 1 - exp(-dt/t_mn)

```
P_scat = 1 - exp(-(dt)/t_mn);
numScat = zeros(1,numTimeStep+1); % just look at scattering of
 electron 1

for n = 0:numTimeStep

    currTime(n+1) = n*dt; % determine current time (ms)

    % Update according to Newton's laws of motion
    % new position = old position + velocity*time

    if n > 0 %update position after t=0

        randVal = rand(numElec,1); % assign scatter probability
        scatter = randVal<=P_scat;
        currVX(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;
        currVY(scatter) = v_th + sqrt(C.kb*Temp/C.m_0)*randn;

        % chance to invert direction when scattering
        randVal = rand(numElec,1);
        invertDir = scatter & (randVal<=0.5);
        currVX(invertDir) = -currVX(invertDir);
        currVY(invertDir) = -currVY(invertDir);

        checkBounce;

    end

    % save previous position to plot trajectory
    prevX(:,n+1) = currX;
    prevY(:,n+1) = currY;

    % make plot

    figure(7)
    for i = 1:nPlottedElec
        plot(prevX(i,:),prevY(i,:))
        hold on
    end
    axis([0 nomRegionL 0 nomRegionW])
    makeBox(B.Left1,B.Right1,B.Top1,B.Bottom1,nomRegionL,nomRegionW);
    makeBox(B.Left2,B.Right2,B.Top2,B.Bottom2,nomRegionL,nomRegionW);
    hold off

end
```



Published with MATLAB® R2021a