Assignment 3

Table of Contents

Matthew Lazarus 100962142

Question 3: Combination of Assignment 1 & 2

In this question, we use the finite difference method to create an electric field for the partciles in the monto carlo simulation.

```
% Clear all previous variables, figures, etc, to ensure that the
 workspace
% is clean.
clear all
clearvars
clearvars -GLOBAL
close all

%Define constants that may need to be used later in the code.
global C
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;             % electron mass
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; %metres (32.1740 ft) per sÂ²

for repeat = 1:1

    %Set the length and width of the grid.
    L=200;
    W=100;
    Lb = 40;
    Wb = 40;

    %Initialize the G,B and conductivity matrices.
    G = sparse(L*W,L*W);
    B=zeros(L*W,1);
    B(1:W,1)=1;
    condMap = zeros(W,L);
```

```
    %Populate the conductivity matrix.
    for lCount = 1:L
        for wCount = 1:W
            if(lCount < L/2 + Lb/2 && lCount > L/2 - Lb/2 &&...
                    (wCount > W-Wb || wCount < Wb))
                condMap( wCount,lCount) = 10^-2;
            else
                condMap(wCount,lCount) = 1;
            end
        end
    end

    % Set the diagonal of the G matrix to 1. This value will be
 overwritten
    % later if it is not a boundary condition.
    for count = 1:L*W
        G(count,count)=1;
    end
```

Loop through rows and columns, if not a boundary case, set the gradient based on the sum of adjacent conductivies.

```
    for col = 1:L
        if(col~=1 &&col~=L)
            for row = 1:W
                n = row + (col -1)*W;
                if(count~=1 && row ~=1 && row~=W)
                    rxBefore = (condMap(row,col) +
 condMap(row,col-1))/2.0;
                    rxAfter = (condMap(row,col) + condMap(row,col
+1))/2.0;
                    ryBefore = (condMap(row,col) +
 condMap(row-1,col))/2.0;
                    ryAfter = (condMap(row,col) + condMap(row
+1,col))/2.0;

                    nyBefore = n-1;
                    nyAfter = n+1;
                    nxBefore = row+(col-2)*W;
                    nxAfter = row+col*W;
                    G(n,n) = -(rxBefore+rxAfter+ryBefore+ryAfter);
                    G(n, nyBefore) =ryBefore;
                    G(n, nyAfter)=ryAfter;
                    G(n, nxBefore)=rxBefore;
                    G(n, nxAfter) =rxAfter;

                elseif(row==1)
                    %Special Case: Bottom of Grid
                    rxBefore = (condMap(row,col) +
 condMap(row,col-1))/2.0;
                    rxAfter = (condMap(row,col) + condMap(row,col
+1))/2.0;
                    ryAfter = (condMap(row,col) + condMap(row
+1,col))/2.0;
```

```
                    nyAfter = n+1;
                    nxBefore = row+(col-2)*W;
                    nxAfter = row+col*W;
                    G(n,n) = -(rxBefore+rxAfter+ryAfter);
                    G(n, nyAfter)=ryAfter;
                    G(n, nxBefore)=rxBefore;
                    G(n, nxAfter) =rxAfter;

                elseif(row==W)
                    %Special Case: Top of Grid
                    rxBefore = (condMap(row,col) +
 condMap(row,col-1))/2.0;
                    rxAfter = (condMap(row,col) + condMap(row,col
+1))/2.0;
                    ryBefore = (condMap(row,col) +
 condMap(row-1,col))/2.0;

                    nyBefore = n-1;
                    nxBefore = row+(col-2)*W;
                    nxAfter = row+col*W;
                    G(n,n) = -(rxBefore+rxAfter+ryBefore);
                    G(n, nyBefore) =ryBefore;
                    G(n, nxBefore)=rxBefore;
                    G(n, nxAfter) =rxAfter;
                end
            end
        end
    end
    V=G\B;

    %Map the voltage to original grid.
    voltMap = zeros(W,L);
    for cols = 1:L
        for rows = 1:W
            n= rows+(cols-1)*W;
            voltMap(rows,cols)=V(n);
        end
    end
```

Find the Electric field knowing $E = -\nabla V$

```
    [Ex, Ey]=gradient(voltMap);
    Ex=-Ex;
    Ey=-Ey;
```

Find the current density knowing $J = \sigma E$

```
    Jx = condMap.*Ex;
    Jy = condMap.*Ey;

    %Sum the currents at both contacts (edges), take the average to
 find
    %the total.
    current1 = sum(Jx(:,1));
```

```
    current2 = sum(Jx(:,L));
    totalCurrent = (current1+current2)/2;

end
```

Monte-Carlo Simulation

Now, use the electric field calculated above to accelerate the particles in the Monte-Carlo Simulatio

```
%Thermal Velocity at 300K:
vth = sqrt(C.kb*300/(0.26*C.m_0));

%Find Mean Free Path
tmn = 0.2*10^-12;
freePath = tmn * vth;

% Set the number of electrons, time step and total time. Initialize
% matrices for the x and y positions, the x and y components of the
% velocity, and the temperature of the system. Column 1 of each
 matrice is
% the previous value, while column 2 is the current value.
numElectrons=20000;
dt = 1e-15; %seconds
nTime = 1e-12; %Simulation length
x = zeros(numElectrons,2); %Position (x)
y = zeros(numElectrons, 2); %Position (y)
vx = zeros(numElectrons, 2); %Velocity (x)
vy = zeros(numElectrons, 2); %Velocity (y)
vTotal = zeros(numElectrons, 2); %Velocity
temperature = zeros(numElectrons,2);
Ix = zeros(numElectrons,2);

% Define two vectors to keep track of the time and distance since the
 last
% scatter, for each electron. Additionally initalize a counter for
 them.
distanceSinceLastScatter = zeros(numElectrons,1);
timeSinceLastScatter = zeros(numElectrons,1);
scatterCount = 0;
```

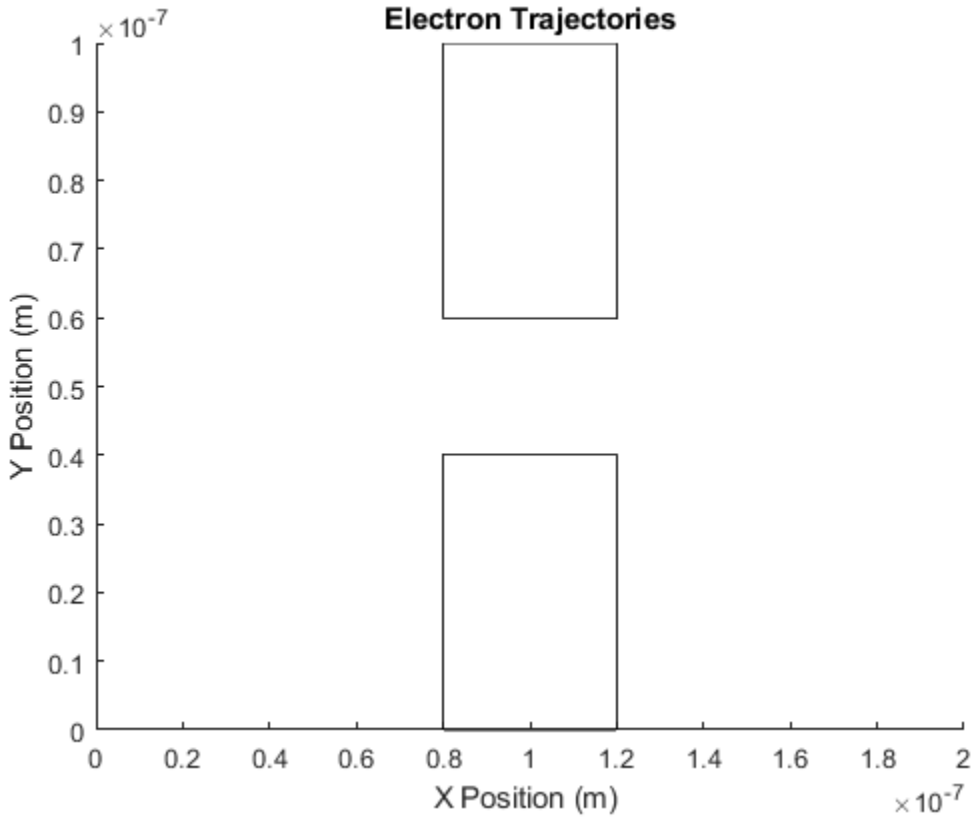
Now, randomly assign initial positions & velocities. Choose each velocity component based off of the Gaussian distribution. Divide by the square root of 2 as there are 2 degrees of freedom. This is derived from equating the acceleration of the electron to its kinetic energy, as seen by the equation below.

$$(1/2)mv^2 = (2/2)kT$$

```
for electronCount = 1:numElectrons
    x(electronCount,2)=rand()*200e-9;
    if(x(electronCount,2)<1.2e-7 && x(electronCount,2)>0.8e-7)
        y(electronCount,2)=rand()*20e-9+0.4e-7;
    else
        y(electronCount,2)=rand()*100e-9;
    end
```

```
    vx(electronCount,2) = vth * randn()/sqrt(2);
    vy(electronCount,2) = vth * randn()/sqrt(2);
    vTotal(electronCount,2)=
 sqrt(vx(electronCount,2)^2+vy(electronCount,2)^2);
end

% Create a figure for the electron trajectories and the temperature of
 the
% system.
figure(1)
title('Electron Trajectories')
xlabel('X Position (m) ')
ylabel('Y Position (m)')
rectangle('Position', [0.8e-7 0 0.4e-7 0.4e-7])
rectangle('Position', [0.8e-7 0.6e-7 0.4e-7 0.4e-7])
axis([0 200e-9 0 100e-9]);
```



The relationship between the electron drift current density and the average carrier density is: $J_d = n * v_{ave}/A$ Where n is the number of electrons.

Therefore, the current is: $I = n * v_{ave}$

```
% Define a vector that will indicate whether an electron crosses a
 horizontal
% boundary. As only 5 electrons will be plotted, if the electron cross
 the
```

```
% boundary, a 1 will be set in the position of the vector that
 corresponds
% to the number of the electron (1-5).
xBreakpoint = zeros(5);
```

A

The plot of the electrons can be seen below. It can be seen that the trajectories are slightly curved because of the acceleration due to the electric field.

Using the probability of scattering equation, seen below,the probability of scattering was determined and compared to a random number between 0 and 1. If the number is less than the probability of scattering, the electron will scatter. This means that it will be given new velocity component values, calculated using the Gaussian distribution as was done above. Lastly, the every time the electron scatters, the time and distance since its last scatter is recorded.

$$P_{scat} = 1 - e^{-dt/\tau_{mn}}$$

```
% Run simulation over time.
for count =    1:2000
 % Run through each electron.
    for c = 1:numElectrons
        if(count~=1)
            % Calculate probability of scattering.
            Pscat = 1-exp(-dt/tmn);

            %  Update the previous positions and velocities.
            vx(c,1)=vx(c,2);
            vy(c,1)=vy(c,2);
            x(c,1)=x(c,2);
            y(c,1)=y(c,2);

            % C: Update the velocities due to the applied voltage
            yE = ceil(y(c,1)/1e-9);
            if(yE<1)
                yE=1;
            elseif(yE>100)
                yE=100;
            end

            xE = ceil(x(c,1)/1e-9);
            if(xE<1)
                xE=1;
            elseif(xE>200)
                xE=200;
            end

            accX  = 1e9*Ex(yE, xE)*C.q_0/(C.m_0); %m/s^2
            accY  = 1e9*Ey(yE, xE)*C.q_0/(C.m_0); %m/s^2
            vx(c,2) = vx(c,2) + accX*dt;
            vy(c,2) = vy(c,2) + accY*dt;

            % Update the current position of the electron.
            x(c,2) = x(c,1) + vx(c, 2)*dt;
```

```
            y(c,2) = y(c,1) + vy(c, 2)*dt;

            %Update time and distance since the last scatter

 distanceSinceLastScatter(c)=distanceSinceLastScatter(c)+sqrt((x(c,2)-
x(c,1))^2+(y(c,2)-y(c,1))^2);
            timeSinceLastScatter(c)=timeSinceLastScatter(c)+dt;


            % Check to see if an electron hit a boundary. If it hit a
            % horizontal boundary, move it to the other side of the
 grid
            % (with the same velocity). If it hit a vertical boundary,
 it
            % should bounce off.
            if(x(c,2)>200e-9)
                x(c,2) = x(c,2)-200e-9;
                if(c<6 && c>0)
                    xBreakpoint(c)=1;
                end
            elseif(x(c,2)<0)
                x(c,2)=x(c,2)+200e-9;
                if(c<6 && c>0)
                   xBreakpoint(c)=1;
                end
            end
            if(y(c,2)>=100e-9)
                vy(c,2) = -vy(c,2);
            elseif(y(c,2)<=0)
                vy(c,2)=-vy(c,2);
            end

            % Check to see if an electron hit a barrier. If it hit, it
            % should bounce off.
            if(x(c,2)<1.2e-7 && x(c,2)>0.8e-7 &&(y(c,2)<0.4e-7 ||
 y(c,2)>0.6e-7))
                %Hit Box. Hit Sides if:
                if((y(c,2-1)<0.4e-7 || y(c,2-1)>0.6e-7))
                    vx(c,2)=-vx(c,2);
                else
                    vy(c,2)=-vy(c,2);
                end
            end

             %Check if the electron scatters
            r=rand();
            if(Pscat>r)
              %Scattering occurs. Update velocity components, then
 save
              % and clear scattering time and distance.
                vx(c,2) = vth * randn()/sqrt(2);
                vy(c,2) = vth * randn()/sqrt(2);

                scatterCount = scatterCount +1;
```

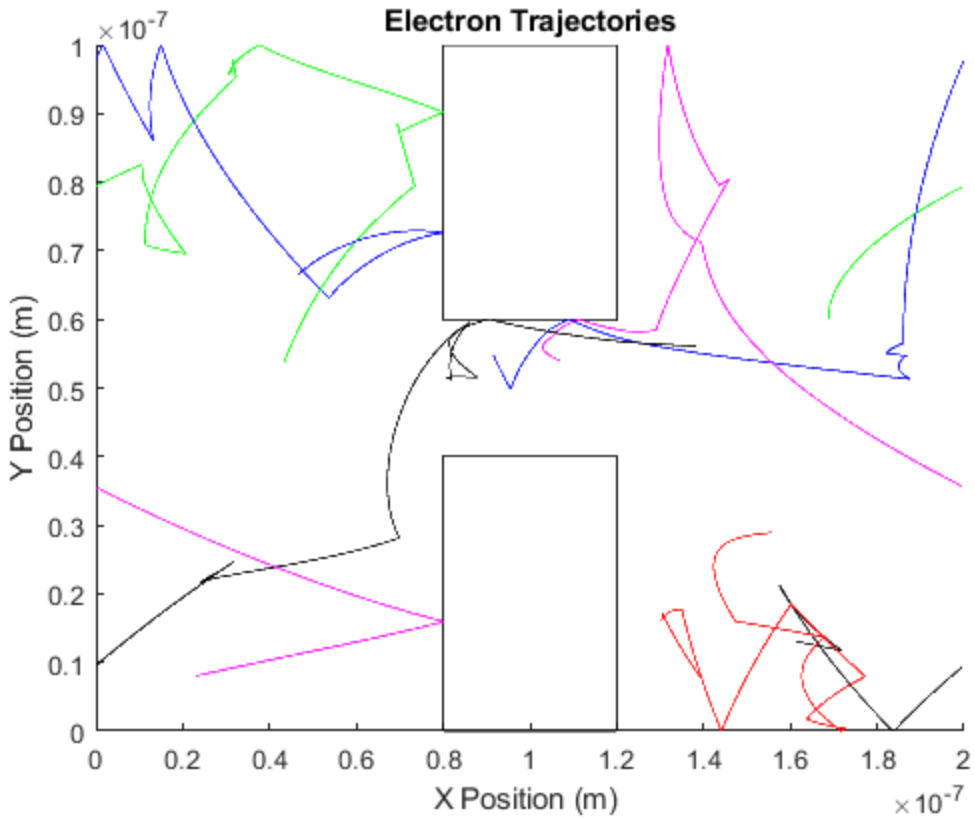
```
                scatterDistances(scatterCount) =
 distanceSinceLastScatter(c);
                scatterTimes(scatterCount)=timeSinceLastScatter(c);
                distanceSinceLastScatter(c)=0;
                timeSinceLastScatter(c)=0;
            end
        end
    end

    if(count>1)
        % Plot the displacement of the electrons in different colours.
        %figure(1)
        hold on
        if(xBreakpoint(1)~=1)
            plot(x(1,1:2),y(1,1:2),'b')
        end
        if(xBreakpoint(2)~=1)
            plot(x(2,1:2),y(2,1:2),'r')
        end
        if(xBreakpoint(3)~=1)
            plot(x(3,1:2),y(3,1:2),'g')
        end
        if(xBreakpoint(4)~=1)
            plot(x(4,1:2),y(4,1:2),'k')
        end
        if(xBreakpoint(5)~=1)
            plot(x(5,1:2),y(5,1:2),'m')
        end
        hold off
    end

    % Reset xBreakpoint.
    xBreakpoint(:)=0;

    % Update the previous and current temperature values. Plot the
 change
    % in temeperature over the step in time.
    temperature(:,1)=temperature(:,2);
    temperature(:,2) = (vx(:,2).^2 + vy(:,2).^2).*((0.26*C.m_0))./
C.kb;

    % Update the current values. Plot the change
    % in current over the step in time.
    Ix(:,1)=Ix(:,2);
    Ix(:,2)= (vx(:,2))*(10^15/(1e-4))*(200e-9*100e-9);
    pause(0.000001)
end
```



Electron Density and Temperature Map

Divide the total grid into 40x40 grid for electron density and velocity (this will give 1600 "boxes"). For each electron, find its corresponding box and sum the number of electrons and total temperature for each box. Ensure no electrons are in the barrier regions.

```
electronCount=zeros(40,40);
electronTemperature=zeros(40,40);
for  eCount = 1:numElectrons
    % Find the box associated with the electron.
    xVal = ceil(40*x(eCount,2)/200e-9);
    yVal = ceil(40*y(eCount,2)/100e-9);
    if(xVal<1)
        xVal=1;
    end
    if(yVal<1)
        yVal =1;
    end
    if(xVal>40)
        xVal = 40;
    end
    if(yVal>40)
        yVal=40;
    end
    %Ensure no electron is within the barriers.
    if((yVal>24 || yVal<17)&&xVal>16&&xVal<25)
```

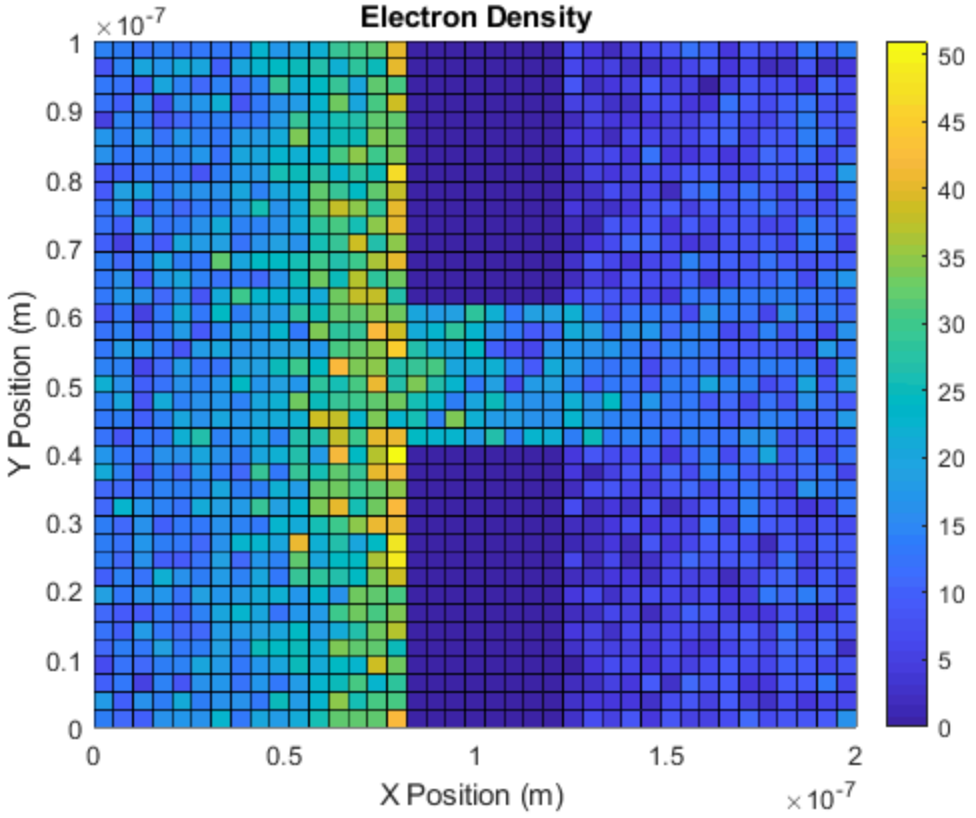
```
        if(xVal==24)
            xVal = 25;
        end
        if(xVal==17)
            xVal = 16;
        end
        if(xVal>17 && xVal <24)
            if(yVal>24)
                yVal = 24;
            elseif(yVal<17)
                yVal=17;
            end
        end
    end

    % Add to the total electron count and total temperature.
    electronCount(yVal,xVal) =electronCount(yVal,xVal)+1;
    electronTemperature(yVal, xVal) = electronTemperature(yVal, xVal)
 + temperature(eCount,2);
end
```

B

Seen below is the electron density plot. It can be seen that the particles gather on the left side of the barrier.

```
% Plot the Electron Density
figure(4)
xAxis = linspace(0,200e-9,40);
yAxis = linspace(0,100e-9,40);
surf(xAxis,yAxis, electronCount);
colorbar;
view(2);
title('Electron Density')
xlabel('X Position (m) ')
ylabel('Y Position (m)')
```



C

In order to make this simulation more accurate, the effect between electrons (forces between electrons) should be take into account.

```
% The mesh size of the
% forward difference method should be increased (to get a more
 accurate E
% field), and the mesh size of the Monte-Carlo Simulation should be
% increased (to get a more accurate position of the particle).
```

Published with MATLAB® R2018a