ELEC4700 Assignment 1

Table of Contents

Reyad ElMahdy 1
101064879 1
Part 2: Collisions with Mean Free Path 3
Part 3: Enhancements 5

Reyad ElMahdy

101064879

```
clc
close all
clear

%& Part 1: Electron Modelling
% Setting the constants to be used in the calculations
mElec = 9.11e-31; % Electron rest mass (kg)
mEff = 0.26*mElec; % Effective mass (kg)
kb = 1.381e-23; % Boltzmann's Constant (J/K)
T = 300; % Room Temperature (K)
L = 200e-9; % Length (m)
W = 100e-9; % Width (m)

% Finding the thermal velocity
Vth = sqrt((2*kb*T)/mEff); %Thermal Velocity in m/s
% Displaying the velocity in km/s on the command window
fprintf('Assuming the temperature T to be 300K the thermal velocity is
 %f km/s. \n', Vth/1000)

% Finding the mean free path
mt = 0.2e-12; % mean time (s)
mfp = Vth*mt; % mean free path (m)
% Displaying the value in nm as the value is too small in meters
fprintf('Assuming the mean time to be 0.2 ps, the mean free path is %f
 nm.\n', mfp*1e9)

% Modeling the electron paths

numPar = 20; % Number of particles

% Assigning particle positions
posX = L.*rand(numPar,2);
posY = W.*rand(numPar,2);
posX(:,1) = posX(:,2);
posY(:,1) = posY(:,2);
```

```
% Assigning particle Directions (and velocity)
angle = (2*pi).*rand(numPar,2);
dirX = Vth*cos(angle);
dirY = Vth*sin(angle);
dirX(:,1) = dirX(:,2);
dirY(:,1) = dirY(:,2);

% Calculating step time
spacialStep = sqrt(L^2+W^2)/100;
stepTime = spacialStep/Vth;

% Calculating Displacement per step
dispX = stepTime*dirX(:,1);
dispY = stepTime*dirY(:,1);

colours = rand(numPar,3);

% Looping through and creating the simulation
for i = 1:1000
    for j = 1:numPar
        % Checking boundary conditions for the right and left
 boundaries of the
        % region
        if (posX(j,1)+dispX(j) > 2e-7)
            posX(j,2) = posX(j,1)+dispX(j)-2e-7;
        elseif (posX(j,1)+dispX(j)<0)
            posX(j,2) = posX(j,1)+dispX(j)+2e-7;
        else
            posX(j,2) = posX(j,1)+dispX(j);
        end

        % Checking conditions for the top and bottom boundaries
        if (posY(j,1)+dispY(j) > 1e-7)|| (posY(j,1)+dispY(j) < 0)
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        else
            posY(j,2) = posY(j,1)+dispY(j);
        end
    end

    % Calculating average Kinetic Energy
    KEmat = 0.5*(mEff*(dirX.^2 + dirY.^2));
    avgKE(i) = sum(KEmat(:,1))/numPar;

    % Calculating the temperature of the semicondctor
    sTemp(i) = avgKE(i)/kb;

    % Generating the plot
    if (i-1 == 0)
        figure(1)
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
        hold on
        title('2-D Particle pathing,Reyad ElMahdy 101064879')
```

```
            xlabel('X-Axis (m)')
            ylabel('Y-Axis (m)')
            xlim([0 200e-9])
            ylim([0 100e-9])
        elseif (i-1 < 1000)
            scatter(posX(:,2),posY(:,2),1,colours(:,1))
            hold on
        else
            scatter(posX(:,2),posY(:,2),1,colours(:,1))
            hold off
        end
        pause(0.001)

        % Updating position vectors between iterations
        posX(:,1) = posX(:,2);
        posY(:,1) = posY(:,2);

        % Storing step times in a time vector
        timeVec(i) = stepTime*i;
    end
    % Plotting the temperature to show that it is constant
    figure(2)
    plot(timeVec,sTemp)
    title({['Semiconductor Temperature'],['Reyad ElMahdy, 101064879']})
    xlabel('Time (seconds)')
    ylabel('Temperature (Kelvin)')
```

Part 2: Collisions with Mean Free Path

```
    % Assigning random velocities to each particle
    Vx = randn(numPar,2)*sqrt((kb*T)/mEff);
    Vy = randn(numPar,2)*sqrt((kb*T)/mEff);

    % Generating the histogram
    V = (Vx(:,1).^2 + Vy(:,1).^2).^0.5;
    figure(3)
    histogram(V,numPar)
    title({['Histogram of Particle Speeds'],['Reyad ElMahdy, 101064879']})
    xlabel('Velocity (m/s)')
    ylabel('Number of Particles')

    dispX = stepTime*Vx(:,1);
    dispY = stepTime*Vy(:,1);

    Pscatter = 1 - exp(-stepTime/mt); % Probability of an electron
     scattering

    % Positions before collision
    collX = posX(:,1);
    collY = posY(:,1);

    collTime = 0;
    numColl = 0;
```

```
sTemp = zeros(1,1000);

for i = 1:1000
    for j = 1:numPar
        % Checking if the electron scatters
        if(rand(1) < Pscatter)
            % Incrementing the collision counter and generating new
            % velocity/displacment values
            numColl=numColl+1;
            angle = 2*pi.*rand(1);
            Vx(j,:) = randn(1)*sqrt((kb*T)/mEff);
            Vy(j,:) = randn(1)*sqrt((kb*T)/mEff);
            dispX(j,:) = stepTime*Vx(j,1);
            dispY(j,:) = stepTime*Vy(j,1);

            % Finding the distance travelled and the time before a
 collision
            dCollX = posX(j,2) - collX(j);
            dCollY = posY(j,2) - collY(j);
            dColl(numColl) = sqrt(dCollX^2+dCollY^2);
            tCollVec(numColl) = abs(stepTime*j - collTime);
            collTime = stepTime*j;

            collX = posX(:,1);
            collY = posY(:,1);
        end

        % Same as Q1
        if (posX(j,1)+dispX(j) > 2e-7)
            posX(j,2) = posX(j,1)+dispX(j)-2e-7;
        elseif (posX(j,1)+dispX(j)<0)
            posX(j,2) = posX(j,1)+dispX(j)+2e-7;
        else
            posX(j,2) = posX(j,1)+dispX(j);
        end
        if (posY(j,1)+dispY(j) > 1e-7)|| (posY(j,1)+dispY(j) < 0)
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        else
            posY(j,2) = posY(j,1)+dispY(j);
        end
    end
    % Calculating kinetic energy
    KEmat = 0.5*(mEff*(Vx.^2 + Vy.^2));
    avgKE(i) = sum(KEmat(:,1))/numPar;
    sTemp(i) = sTemp(i)+avgKE(i)/kb;
    sTempAvg(i) = sTemp(i)/i;
    timeVec(i) = stepTime*i;

    if (i-1 == 0)
        figure(4)
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
        hold on
```

```
        title('2-D Particle pathing, Reyad ElMahdy 101064879')
        xlabel('X-Axis (m)')
        ylabel('Y-Axis (m)')
        xlim([0 200e-9])
        ylim([0 100e-9])
    elseif (i-1 < 1000)
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
    else
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
        hold off
    end
    pause(0.001)

    % Updating position vectors between iterations
    posX(:,1) = posX(:,2);
    posY(:,1) = posY(:,2);

    % Storing step times in a time vector
    timeVec(i) = stepTime*i;
end
% Plotting the temperature
figure(5)
plot(timeVec,sTempAvg)
title({['Semiconductor Temperature'],['Reyad ElMahdy, 101064879']})
xlabel('Time (seconds)')
ylabel('Temperature (Kelvin)')

% Calclating the MFP
dTotal = sum(dColl);
MFP = dTotal/numColl;
fprintf('The MFP of the simulation is %f nm \n', MFP*1e9)
% Calculating the mean time between collisions
timeTotal = sum(timeVec);
meanTime = timeTotal/numColl;
fprintf('The mean time between collisions of the simulation is %f ps
 \n', meanTime*1e12)
```

Part 3: Enhancements

```
Vx = randn(numPar,2)*sqrt((kb*T)/mEff);
Vy = randn(numPar,2)*sqrt((kb*T)/mEff);
V = (Vx(:,1).^2 + Vy(:,1).^2).^0.5;

dispX = stepTime*Vx(:,1);
dispY = stepTime*Vy(:,1);
%Ensure no particles spawn inside the boxes
for i = 1:numPar
    while ((posX(i,2) > 0.8e-7 && posX(i,2) < 1.2e-7) && (posY(i,2) >
 0.6e-7 ||  posY(i,2) < 0.4e-7))
        posX(i,2) = L.*rand(1,1);
        posY(i,2) = W.*rand(1,1);
    end
end
```

```
for i = 1:1000
    for j = 1:numPar
        % Checking if the electron scatters
        if(rand(1) < Pscatter)
            % Incrementing the collision counter and generating new
            % velocity/displacment values
            numColl=numColl+1;
            angle = 2*pi.*rand(1);
            Vx(j,:) = randn(1)*sqrt((kb*T)/mEff);
            Vy(j,:) = randn(1)*sqrt((kb*T)/mEff);
            dispX(j,:) = stepTime*Vx(j,1);
            dispY(j,:) = stepTime*Vy(j,1);

            % Finding the distance travelled and the time before a
 collision
            dCollX = posX(j,2) - collX(j);
            dCollY = posY(j,2) - collY(j);
            dColl(numColl) = sqrt(dCollX^2+dCollY^2);
            tCollVec(numColl) = abs(stepTime*j - collTime);
            collTime = stepTime*j;

            collX = posX(:,1);
            collY = posY(:,1);
        end

        % More boundary conditions need to be added to simulate the
 boxes
        % in part 3
        if (posX(j,1)+dispX(j) > 2e-7)
            posX(j,2) = posX(j,1)+dispX(j)-2e-7;
        elseif (posX(j,1)+dispX(j)<0)
            posX(j,2) = posX(j,1)+dispX(j)+2e-7;
        elseif (posX(j,1) >= 0 && posX(j,1) <= 0.8e-7) && (posY(j,1)
>= 0 && posY(j,1) <= 0.4e-7) && (posX(j,1)+dispX(j,1) >= 0.8e-7)
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1) + dispX(j);
        elseif (posX(j,1) >= 0 && posX(j,1) <= 0.8e-7) && (posY(j,1)
>= 0.6e-7 && posY(j,1) <= 1e-7) && (posX(j,1)+dispX(j) >= 0.8e-7)
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1) + dispX(j);
        elseif (posX(j,1) >= 1.2e-7 && posX(j,1) <= 2e-7) &&
(posY(j,1) >= 0 && posY(j,1) <= 0.4e-7) && (posX(j,1)+dispX(j) <=
1.2e-7)
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1) + dispX(j);
        elseif (posX(j,1) >= 1.2e-7 && posX(j,1) <= 2e-7) &&
(posY(j,1) >= 0.6e-7 && posY(j,1) <= 1e-7) && (posX(j,1)+dispX(j) <=
1.2e-7)
            dispX(j) = -dispX(j);
            posX(j,2) = posX(j,1) + dispX(j);
        else
            posX(j,2) = posX(j,1)+dispX(j);
        end
```

```
        if (posY(j,1)+dispY(j) > 1e-7) || (posY(j,1)+dispY(j) < 0)
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        elseif(posX(j,1)>=0.8e-7 && posX(j,1) <= 1.2e-7) && (posY(j,1)
>= 0.4e-7 && posY(j,1) <= 0.6e-7) && (posY(j,1)+dispY(j) <= 0.4e-7)
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        elseif(posX(j,1)>=0.8e-7 && posX(j,1) <= 1.2e-7) && (posY(j,1)
>= 0.4e-7 && posY(j,1) <= 0.6e-7) && (posY(j,1)+dispY(j) >= 0.6e-7)
            dispY(j) = -dispY(j);
            posY(j,2) = posY(j,1)+dispY(j);
        else
            posY(j,2) = posY(j,1)+dispY(j);
        end
    end
    % Calculating kinetic energy
    KEmat = 0.5*(mEff*(Vx.^2 + Vy.^2));
    avgKE(i) = sum(KEmat(:,1))/numPar;
    sTemp(i) = sTemp(i)+avgKE(i)/kb;
    sTempAvg(i) = sTemp(i)/i;
    timeVec(i) = stepTime*i;

    if (i-1 == 0)
        figure(6)
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
        hold on

        % Plotting the boxes
        plot([0.8e-7,0.8e-7],[0,0.4e-7],'k',[0.8e-7,1.2e-7],
[0.4e-7,0.4e-7],'k',[1.2e-7,1.2e-7],[0.4e-7,0],'k',[0.8e-7,1.2e-7],
[0,0],'k')
        plot([0.8e-7,0.8e-7],[1e-7,0.6e-7],'k',[0.8e-7,1.2e-7],
[0.6e-7,0.6e-7],'k',[1.2e-7,1.2e-7],[0.6e-7,1e-7],'k',[0.8e-7,1.2e-7],
[1e-7,1e-7],'k')

        title('2-D Particle pathing, Reyad ElMahdy 101064879')
        xlabel('X-Axis (m)')
        ylabel('Y-Axis (m)')
        xlim([0 200e-9])
        ylim([0 100e-9])
    elseif (i-1 < 1000)
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
    else
        scatter(posX(:,2),posY(:,2),1,colours(:,1))
        hold off
    end
    pause(0.001)

    % Updating position vectors between iterations
    posX(:,1) = posX(:,2);
    posY(:,1) = posY(:,2);

    % Storing step times in a time vector
    timeVec(i) = stepTime*i;
```

```
end

% Electron Density Map
figure(7)
hist3([posX(:,1),posY(:,1)],[10,20])
title('Electron density map, Reyad ElMahdy 101064879')
zlabel('Number of Particles')

% Temperature Map
kEtemp = (mEff.*V.^2)/2;
temp = kEtemp./kb;

% Scaling and rounding positions to the nearest nm
posXnm = round(posX(:,1).*1e9, -1)./10;
posYnm = round(posY(:,1).*1e9, -1)./10;

tempArr = zeros(round(W/1e-9),round(L/1e-9));

for i = 1:numPar
    posXnm = round(posX(i,1).*1e9,-1)/10;
    posYnm = round(posY(i,1).*1e9,-1)/10;

    if(posXnm <= 0 || posYnm <= 0)
        posXnm = 1;
        posYnm = 1;
    end
    tempArr(posXnm,posYnm) = tempArr(posXnm,posYnm) + temp(i);
end

matrix = hist3([posX(:,1),posY(:,1)],[10,20]);

for i = 1:round(W/1e-9)/10
    for j = 1:round(L/1e-9)/10
        if(matrix(i,j) == 0)
            tempArr(i,j) = 0;
        else
            tempArr(i,j) = tempArr(i,j)/matrix(i,j);
        end
    end
end

figure(8)
bar3(tempArr)
title('Temperature map, Reyad ElMahdy 101064879')
zlabel('Temperature (Kelvin)')
```

Published with MATLAB® R2020b