Jalil (Rohana) Aalab #100995788
ELEC 4700 Assignment 1 Feb/4/2018

Table of Contents

Part 1 : Electron Modelling

This first section covers defining variables. The physical constants that will used later are defined here.

```
Mo = 9.109 * (10^-31);
% electron rest mass in kilograms
Mn = 0.26 * Mo;
% effective electron mass in kilograms
L = 200 * (10^-9);
% length of region in metres
W = 100 * (10^-9);
% width of region in metres
T = 300;
% Temperature in Kelvin
Tmn = 0.2 * (10^-12);
% Mean time of collisions between particles in sec
dt = 5 * (10^-15);
% time step in seconds
TStop = 1000 * dt;
% stop simulation after 1000 time steps
B = 1.38 * (10^-23);
% Boltzmann constant
old_t = 0;
% This will be the time before the timestep
old_temp_test = Temp_test;
% This will be the old calculated system temp

Temp_test = 300;
% This tests to see if the temperature remains constant
% and is set to 300K to check if the system remains at
% this temperature over time.
% P is the position matrix of 1000 particles. The first row contains
 the
% x-coordinates of the points and the second row has the y-
coordinates. The
% coordinates are scaled from a random generator to be within the
 limits of
% the region
p = [zeros(1,1000); zeros(1,1000)];
for i = 1 : 1 : 1000
    p(1,i) = rand * 100 * (10^-9);
    p(2,i) = rand * 100 * (10^-9);
end

% Old P will contain the old position coordinates for the points prior
% to them being updated at every timestep.
old_p = [zeros(1,1000); zeros(1,1000)];
old_p = p;

% Polar coordinates are being used, and theta is the angle that
 determines
% the direction of the path of the electron. The values are chosen
 from a
% uniformly distributed random generator and are scaled between 0 and
 2 pi
% radians.
theta = [zeros(1,1000)];
for a = 1 : 1 : 1000
    theta(a) = rand * (2*pi);
end

% The elements of the bounce vector correspond to each particle; the
 value
% of each element is 1 (TRUE) or 0 (FALSE). The value changes when it
 is
% detected that the particle needs to bounce.
bounce = [ones(1,1000)];

% The cross and cross1 vectors' elements correspond to each particle
 and
% will be used to determine if the right and left boundaries have been
% crossed. The values are each 1 or 0.
cross = [zeros(1,1000)];
cross1 = [zeros(1,1000)];
```

```
Undefined function or variable 'Temp_test'.

Error in assignment1 (line 26)
old_temp_test = Temp_test;
```

Thermal Velocity

From the kinetic theory of gases, the kinetic energy is related to temperature and both mass and velocity through the formula below, where m is mass, v is velocity, k is Boltzmann’s constant and T is temperature.

(1/2)m(v^2) = 1/2 * kT (per degree of freedom) This simulation modelled electron movement in two dimensions, namely in the x and y directions, and therefore had two degrees of freedom. This changed the formula to: (1/2)m(v^2) = (1/2*2) kT = kT The thermal velocity is calculated by rearranging the above formula: Vth = sqrt(2mv/m) The thermal velocity calculation used the above formula while assuming T = 300 K, and an effective electron mass of 2.36834 x 10-31 kilograms. Therefore, Vth = sqrt((2k(300))/((2.36834 x 10-31))) = 1.8698 x 105 m/s

```
Vth = sqrt((B * T * 2)/ Mn);
% Thermal Velocity in metres per second
```

Mean Free Path

The mean free path is defined as the average distance a particle can travel undisturbed prior to a collision with another particle. It is calculated as the product of the thermal velocity and the mean time between collisions. Therefore, MFP = Vth * Tmn. The velocities of all the particles were fixed at the thermal velocity determined above, and the mean collision time was given as Tmn = 0.2 x 10-12 s Therefore, MFP = Vth x Tmn = (1.8698 x 105 m/s) x (0.2 x 10-12 s)= 2.6443 x 10-8 m

```
Mean_path = Tmn * Vth;
% Mean Free Path in metres
rho = Vth * dt;
% Polar coordinates are used to determine the magnitude of
% distance and its angular direction. Rho is the magnitude
% of the distance in metres.
```

2-D Plot

This section covers setting up the plot. The plot has two graphs on one figure; the electron model movement and the average temperature.

```
figure(1);
subplot(2,1,1)
axis([0 L 0 W]);
hold on;
title('Model Electron Movement');

subplot(2,1,2)
hold on;
title('Temperature over Time');
xlabel('Time (sec)');
ylabel('Temperature (Kelvin)');

    for t = 0 : dt : (TStop)
         % goes up to max time steps
      for k = 1 : 1 : 7
        % Sets up the different coloured lines
          if k == 1
           c = 'y';
        else
            if k == 2
               c = 'm';
            else
                if k == 3
```

```
                    c = 'c';
                 else
                     if k == 4
                        c = 'r';
                     else
                         if k == 5
                            c = 'g';
                         else
                             if k == 6
                                c = 'b';
                             else
                                 if k == 7
                                    c = 'k';
                            end
                        end
                    end
                end
            end
            end
        end

        subplot(2,1,1)
        % Now we fix the case that the line crosses the left/right
        % boundaries and we don't want there to be a big line across
the
        % graph from the previous and new points. This situation has
been
        % called 'cross' if the right is crossed, and 'cross1' if the
        % left is crossed. They are both vectors that are initially 0,
and
        % if the situation changes they are called TRUE and the old
point
        % is moved from the previous side to the new side so that when
a
        % line is placed between the old and new points it doesn't
cross
        % the whole page.
        if cross(k) == true
            old_p(1,k) = 0.004 * (10^-7);
        end
        if cross1(k) == true
            old_p(1,k) = 1.996 * (10^-7);
        end
        % The line is made from the old to the new positions
        line([old_p(1,k) p(1,k)], [old_p(2,k) p(2,k)], 'Color', c);
      end

      old_p = p;
      % The current positions become the old positions

      % This loop updates the positions of the x and y coordinates
      for e = 1 : 1 : 1000
          p(1,e) = p(1,e) + (rho * cos(theta(e)));
          % position gets updated
```

```
            if bounce(e) == 1
                p(2,e) = p(2,e) + (rho * sin(theta(e)));
            else
                p(2,e) = p(2,e) - (rho * sin(theta(e)));
                % opposite vertical way
            end
            pause(dt);
        end

        % This loop determines if the right and left boundaries are
    crossed
        for d = 1 : 1 : 1000
            % if the line goes past the right boundary
            if p(1,d) > (1.995 * 10^-7)
                % is the x-coordinate past the right?
                p(1,d) = 0.005 * (10^-7);
                 % bring it to the left side
                cross(d) = true;
            else
                cross(d) = false;
            end

            % if the line goes past the left boundary
            if p(1,d) < (0.005 * 10^-7)
                % is the x-coordinate past the left?
                p(1,d) = 1.995 * (10^-7);
                % bring it to the right side
                cross1(d) = true;
            else
                cross1(d) = false;
            end
        end

        for b = 1 : 1 : 1000
            if p(2,b) < (0.005 * (10^-7))
                 % if line hits bottom
                bounce(b) = bounce(b) * (-1);
                % make it go the other way
            end
        end

            for m = 1 : 1 : 1000
                % if line hits top
                if (1 * (10^-7)) - (p(2,m)) < (0.005 * (10^-7))
                    bounce(m) = bounce(m) * (-1);
                    % make it go the other way
                end
            end
        % Checks if temperature is constant
        Temp_test =((Mn * ((Vth*Vth))) / (B * 2));
        subplot(2,1,2)
        line([old_t t], [old_temp_test Temp_test], 'Color', 'r');
        old_t = t;
        old_temp_test = Temp_test;
        end
```

Part 2:Collisions with Mean Free Path (MFP)

The physical constants are defined here.

```
Mo = 9.109 * (10^-31);
% electron rest mass in kilograms
Mn = 0.26 * Mo;
% effective electron mass in kilograms
L = 200 * (10^-9);
% length of region in metres
W = 100 * (10^-9);
% width of region in metres
T = 300;
% Temperature in Kelvin
Tmn = 0.2 * (10^-12);
% Mean time of collisions between particles in sec
dt = 5 * (10^-15);
% time step in seconds
TStop = 1000 * dt;
% stop simulation after 1000 time steps
B = 1.38 * (10^-23);
% Boltzmann constant
old_t = 0;
% This will be the time before the timestep
old_temp_test = Temp_test;
% This will be the old calculated system temp
Temp_test = 300;

Pscat = 1 - exp((dt * (-1)) / Tmn);
% Probability of scattering

Max_Bolt = (4*pi)*((Mn/(2*pi*B*T))^(1.5))*((Vth*Vth)*(exp(-Mn*Vth*Vth/
(2*B*T))));
% Assigning random velocities
rand_v = [zeros(1,1000)];
for h = 1 : 1 : 1000
    rand_v(h) = (4*pi)*((Mn/(2*pi*B*T))^(1.5))*((h*h)*(exp(-Mn*h*h/
(2*B*T)))) * Vth;
end

rand_rho = [zeros(1,1000)];
for h1 = 1 : 1 : 1000
    rand_rho(h1) = rand_v(h1) * dt;
end

cross = [zeros(1,1000)];
cross1 = [zeros(1,1000)];

old_p = [zeros(1,1000); zeros(1,1000)];

p = [zeros(1,1000); zeros(1,1000)];
```

```
for i = 1 : 1: 1000
    p(1,i) = rand * 100 * (10^-9);
    p(2,i) = rand * 100 * (10^-9);
end

old_p = p;

theta = [zeros(1,1000)];
% Polar coordinates are used; theta is the direction
for a = 1 : 1 : 1000
    theta(a) = rand * (2*pi);
    % Sets up a random angle of direction for each particle
end

scatter = [zeros(1,1000)];

old_avg_v = mean(rand_rho);

bounce = [ones(1,1000)];
% Setting up the plot
figure(2);
subplot(3,1,1)
axis([0 L 0 W]);
hold on;
title('Model Electron Movement');

subplot(3,1,2)
hold on;
title('Avg Temp');
xlabel('Time (sec)');
ylabel('Temperature (Kelvin)');
```

Histogram

This produces a histogram of the initial velocities.

```
subplot(3,1,3)
hist(rand_rho,100);
title('Velocity Histogram');
xlabel('Velocity');
ylabel('#Particles');

old_t = 0;
```

2-D Plot

```
    for t = 0 : dt : (TStop)
        % goes up to max time steps
      for k = 1 : 1 : 7
        % Sets up the different coloured lines
          if k == 1
           c = 'y';
```

```
  else
      if k == 2
         c = 'm';
      else
          if k == 3
             c = 'c';
          else
              if k == 4
                 c = 'r';
              else
                  if k == 5
                     c = 'g';
                  else
                      if k == 6
                         c = 'b';
                      else
                          if k == 7
                             c = 'k';
                     end
                 end
             end
         end
     end
     end
  end
  subplot(3,1,1)
  if cross(k) == true
      old_p(1,k) = 0.004 * (10^-7);
  end
  if cross1(k) == true
      old_p(1,k) = 1.996 * (10^-7);
  end
  line([old_p(1,k) p(1,k)], [old_p(2,k) p(2,k)], 'Color', c);
end

  old_p = p;

% Scatter Check
for sc = 1 : 1 : 1000
    if scatter(sc) == true
        rand_rho(sc) = rand * 100 * (10^4) * dt;
        theta(sc) = rand * (2*pi);
    end
end

for e = 1 : 1 : 1000
   p(1,e) = p(1,e) + (rand_rho(e) * cos(theta(e)));
   % position gets updated
    if bounce(e) == 1
        p(2,e) = p(2,e) + (rand_rho(e) * sin(theta(e)));
    else
        p(2,e) = p(2,e) - (rand_rho(e) * sin(theta(e)));
        % opposite vertical way
    end
    pause(dt);
end

for e1 = 1 : 1 : 1000
    if Pscat > rand
        scatter(e1) = true;
    else
        scatter(e1) = false;
    end
end

for d = 1 : 1 : 1000
    % if the line goes past the right boundary
    if p(1,d) > (1.995 * 10^-7)
        % is the x-coordinate past the right?
        p(1,d) = 0.005 * (10^-7);
        % bring it to the left side
        cross(d) = true;
    else
        cross(d) = false;
    end

    % if the line goes past the left boundary
    if p(1,d) < (0.005 * 10^-7)
        % is the x-coordinate past the left?
        p(1,d) = 1.995 * (10^-7);
        % bring it to the right side
        cross1(d) = true;
    else
        cross1(d) = false;
    end
end

for b = 1 : 1 : 1000
    if p(2,b) < (0.005 * (10^-7))
        % if line hits bottom
        bounce(b) = bounce(b) * (-1);
        % make it go the other way
    end
end

    for m = 1 : 1 : 1000
        if (1 * (10^-7)) - (p(2,m)) < (0.005 * (10^-7))
            % if line hits top
            bounce(m) = bounce(m) * (-1);
            % make it go the other way
        end
    end

avg_v = mean(rand_rho)/dt;
Temp_test = (Mn * Vth * Vth) / (B * 2);
% Checks if temperature is constant
subplot(3,1,2)
line([old_t t], [old_avg_v avg_v], 'Color', 'r');
```

```
        old_avg_v = avg_v;
        old_t = t;
    end
```

Part 3: Enhancements

2-D Plot

```
Mo = 9.109 * (10^-31);
% electron rest mass in kilograms
Mn = 0.26 * Mo;
% effective electron mass in kilograms
L = 200 * (10^-9);
% length of region in metres
W = 100 * (10^-9);
% width of region in metres
T = 300;
% Temperature in Kelvin
Tmn = 0.2 * (10^-12);
% Mean time of collisions between particles in sec
dt = 5 * (10^-15);
% time step in seconds
TStop = 1000 * dt;
% stop simulation after 1000 time steps
B = 1.38 * (10^-23);
% Boltzmann constant
Vth = sqrt(B * T/ Mn);
% Thermal Velocity in metres per second
Mean_path = Tmn * Vth;
% Mean Free Path in metres
rho = Vth * dt;
% Gives the distance between each timestep of new point
Temp_test_old = 300;
% This tests to see if the temperature remains constant
t_old = 0;
% This is the old time prior to the new timestep

x1 = (0.8 * (10^-7));
x2 = (1.2 * (10^-7));
y1 = (0.4 * (10^-7));
y2 = (0.6 * (10^-7));
y3 = (1 * (10^-7));
% These are coordinates of the bottle neck boundaries

Pscat = 1 - exp((dt * (-1)) / Tmn);
% Probability of scattering
Max_Bolt = (sqrt(Mn / (2 * pi * B * T))) * (exp((-Mn * Vth * Vth) / (2
 * B * T)));

% Assigning random velocities
rand_v = [zeros(1,1000)];
for h = 1 : 1 : 7
```

```
    rand_v(h) = (4*pi)*((Mn/(2*pi*B*T))^(1.5))*((h*h)*(exp(-Mn*h*h/
(2*B*T)))) * Vth;
end

rand_rho = [zeros(1,1000)];
for h1 = 1 : 1 : 7
    rand_rho(h1) = rand_v(h1) * dt;
end

cross = [zeros(1,1000)];
cross1 = [zeros(1,1000)];

p = [zeros(1,1000); zeros(1,1000)];
for i = 1 : 1: 1000
    p(1,i) = rand * 100 * (10^-9);
    while (p(1,i) >= x1) && (p(1,i) <= x2)
        p(1,i) = rand * 100 * (10^-9);
    end
    p(2,i) = rand * 100 * (10^-9);
end

old_p = [zeros(1,1000); zeros(1,1000)];
old_p = p;

% Polar coordinates are used; theta is the direction
theta = [zeros(1,1000)];
for a = 1 : 1 : 7
    theta(a) = rand * (2*pi);
    % Sets up a random angle of direction for each particle
end

% The scatter vector has values either 0 or 1 to determine
% if the electron path should scatter
scatter = [zeros(1,1000)];

% Setting up the plot
bounce = [ones(1,1000)];
% The xbounce vector has values corresponding to the particles. It is
% either 1 or 0 if the particle bounces off the bottleneck areas or
 not
xbounce = [ones(1,1000)];
figure(3);
subplot(3,1,1)
axis([0 L 0 W]);
% Setting up boxes in plot
line([x1 x2], [y1 y1]);
line([x1 x2], [y2 y2]);
line([x1 x1], [0 y1]);
line([x2 x2], [0 y1]);
line([x1 x1], [y2 y3]);
line([x2 x2], [y2 y3]);
hold on;
title('Model Electron Movement');
```

```
for t = 0 : dt : (TStop)
    % goes up to max time steps
  for k = 1 : 1 : 7
       % Sets up the different coloured lines
    if k == 1
       c = 'y';
    else
        if k == 2
           c = 'm';
        else
            if k == 3
               c = 'c';
            else
                if k == 4
                   c = 'r';
                else
                    if k == 5
                       c = 'g';
                    else
                        if k == 6
                           c = 'b';
                        else
                            if k == 7
                               c = 'k';
                       end
                   end
               end
           end
       end
       end
    end

    subplot(2,1,1)
    if cross(k) == true
        old_p(1,k) = 0.004 * (10^-7);
    end
    if cross1(k) == true
        old_p(1,k) = 1.996 * (10^-7);
    end
    line([old_p(1,k) p(1,k)], [old_p(2,k) p(2,k)], 'Color', c);
  end

     old_p = p;

      for sc = 1 : 1 : 1000
          % Scatter Check
      if scatter(sc) == true
          rand_rho(sc) = rand * 100 * (10^4) * dt;
          theta(sc) = randi([1 179]);
      end
      end


  for n = 1 : 1 : 1000
```

```
        if xbounce(n) == 1
          p(1,n) = p(1,n) + (rand_rho(n) * cos(theta(n)));
          % position gets updated
        else
          p(1,n) = p(1,n) - (rand_rho(n) * cos(theta(n)));
          % position gets updated
        end
    end

    for e = 1 : 1 : 1000
        if bounce(e) == 1
            p(2,e) = p(2,e) + (rand_rho(e) * sin(theta(e)));
        else
            p(2,e) = p(2,e) - (rand_rho(e) * sin(theta(e)));
            % opposite vertical way
        end
        pause(dt);
    end

      for e1 = 1 : 1 : 1000
        if Pscat > rand
            scatter(e1) = true;
        else
            scatter(e1) = false;
        end
      end

   for d = 1 : 1 : 1000
        % if the line goes past the right boundary
        if p(1,d) > (1.995 * 10^-7)
            % opposite vertical way
            p(1,d) = 0.005 * (10^-7);
            % bring it to the left side
            cross(d) = true;
        else
            cross(d) = false;
        end

        % if the line goes past the left boundary
        if p(1,d) < (0.005 * 10^-7)
            % is the x-coordinate past the left?
            p(1,d) = 1.995 * (10^-7);
            % bring it to the right side
            cross1(d) = true;
        else
            cross1(d) = false;
        end
    end

    for q1 = 1 : 1 : 1000
        if (x1 - p(1,q1)) < (0.005 * (10^-7)) && (p(2,q1) < y1)
            xbounce(q1) = xbounce(q1) * (-1);
        end
    end
```

```
        for q2 = 1 : 1 : 1000
            if (x1 - p(1,q2)) < (0.005 * (10^-7)) && (p(2,q2) > y2)
                xbounce(q2) = xbounce(q2) * (-1);
            end
        end

        for q3 = 1 : 1 : 1000
            if (p(1,q3) - x2) > (0.005 * (10^-7)) && (p(2,q3) < y1)
                xbounce(q3) = xbounce(q3) * (-1);
            end
        end

          for q4 = 1 : 1 : 1000
            if (p(1,q4) - x2) > (0.005 * (10^-7)) && (p(2,q4) > y2)
                xbounce(q4) = xbounce(q4) * (-1);
            end
        end


        for b = 1 : 1 : 1000
            if p(2,b) < (0.005 * (10^-7))
                % if line hits bottom
                bounce(b) = bounce(b) * (-1);
                % make it go the other way
            end
        end

             for f = 1 : 1 : 1000
            if (((p(2,f)) - (0.4 * (10^-7))) < (0.005 * (10^-7))) &&
(p(1,f) >= x1) && (p(1,f) <= x2)
                bounce(f) = bounce(f) * (-1);
                % make it go the other way
            end
             end

            for m = 1 : 1 : 1000
                if (1 * (10^-7)) - (p(2,m)) < (0.005 * (10^-7))
                    % if line hits top
                    bounce(m) = bounce(m) * (-1);
                    % make it go the other way
                end
            end

            for g = 1 : 1 : 1000
                if ((0.6 * (10^-7)) - (p(2,g)) < (0.005 * (10^-7))) &&
(p(1,g) >= x1) && (p(1,g) <= x2)
                    bounce(g) = bounce(g) * (-1);
                     % make it go the other way
                end
            end

        t_new = t;
        Temp_test_new = (Mn * Vth * Vth) / (B * 2);
```

```
     % This plots the temperature over time, updating time and
temperature
     subplot(2,1,2)
     line([t_old t_new], [Temp_test_old
Temp_test_new], 'Color', 'r');
     Temp_test_old = Temp_test_new;
     t_old = t_new;

   % This is the new idea for the temperature map
   % h = histogram2(x,y,'Display Style','tile','ShowEmptyBins','on');

   end
```

Final Electron Position Density Map

This is the electron density map of the figure.

```
subplot(3,1,2);
h = histogram2(p(1,:), p(2,:), [20 10]);
```

Final Temperature Plot

This is the temperature map of the figure.

```
subplot(3,1,3);
h = histogram2(rand_v(1,:),[200 100],'FaceColor','flat');
 colorbar
```

Published with MATLAB® R2017b