ELEC 4700 Jalil (Rohana) Aalab #100995788 Assignment 3

Table of Contents

This is the third assignment for ELEC 4700

Part 1 A

This section calculates the electric on the electrons when the voltage is 0.1 V applied across the x dimension. The electric field is the negative vector differential of electric potential, but can also be defined as E = V/d, producing electric field in units of volts per metre.

```
Vx = 0.1;
d = 2 * (10^-7);
E = Vx / d;
fprintf('Therefore, the electric field is %d V/m\n', E);
```

```
Therefore, the electric field is 5.000000e+05 V/m
```

Part 1 B

This section finds the force on each electron. Electric force can be defined as F = Eq.

```
q = -(1.602 * (10^-19));
F = E * q;
fprintf('Therefore, the electric force is %d N\n', F);
```

```
Therefore, the electric force is -8.010000e-14 N
```

Part 1 C

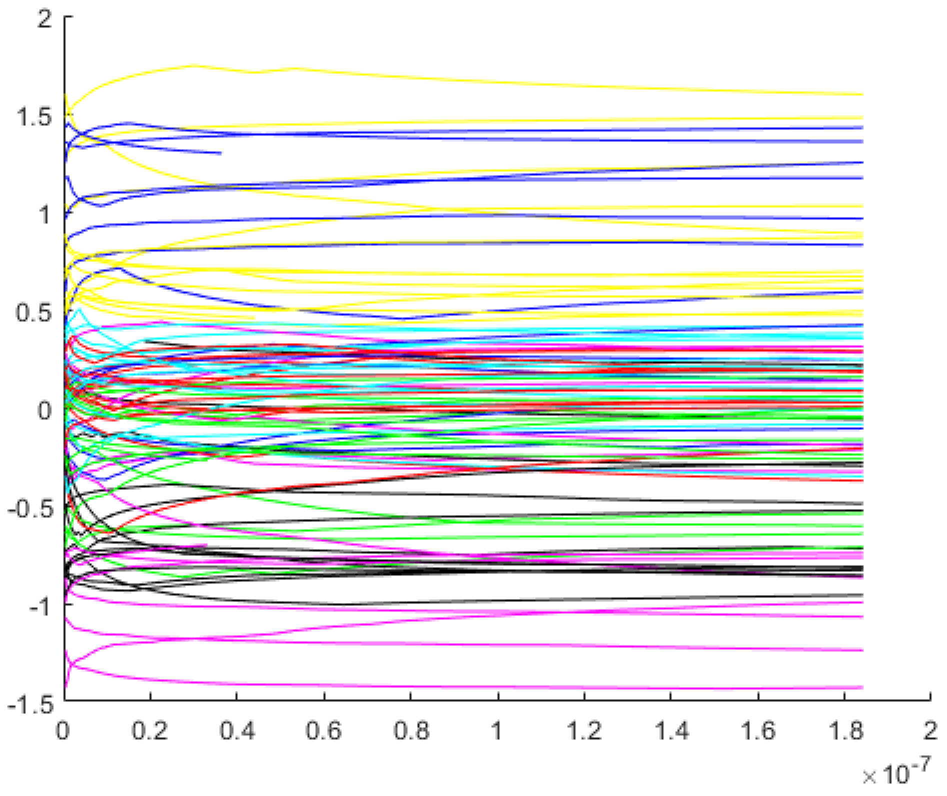
This code finds the electron acceleration and plots their trajectories. Given that F = ma, the acceleration 'a' is a = F/m. This code functions in an odd manner in that the first figure produces inaccurate particle movement but closing the first figure opens a second figure where the particles correctly move in curved trajectories across the x dimension.

```
Mo = 9.109 * (10^-31);
Mn = 0.26 * Mo;
a = F / Mn;
fprintf('Therefore, the acceleration is %d m/s/s\n', a);
% Trajectories based on code from assignment 1 are below
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
Temp_test = 300;
old_temp_test = Temp_test;
% This will be the old calculated system temp



% 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)];
Vth = sqrt((B * T * 2)/ Mn);
Mean_path = Tmn * Vth;
rho = Vth * dt;

figure(1);
axis([0 L 0 W]);
hold on;
title('Model Electron Movement With Electric Field');

x = 0;
y = 0;
    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
        % 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
           x = rho*(cos(theta(e)));
           y = rho*(sin(theta(e)));
           x = x + (((F/Mn)*dt) * (10^-5));
          rho = sqrt((x^2) + (y^2));
          theta(e) = atan(y/x);
          p(1,e) = p(1,e) + (Vx*(p(1,e)));
          % position gets updated
          if bounce(e) == 1
            p(2,e) = p(2,e) + y;
          else
            p(2,e) = p(2,e) - y;
            % 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
    end
```

Therefore, the acceleration is -338211574351655552 m/s/s

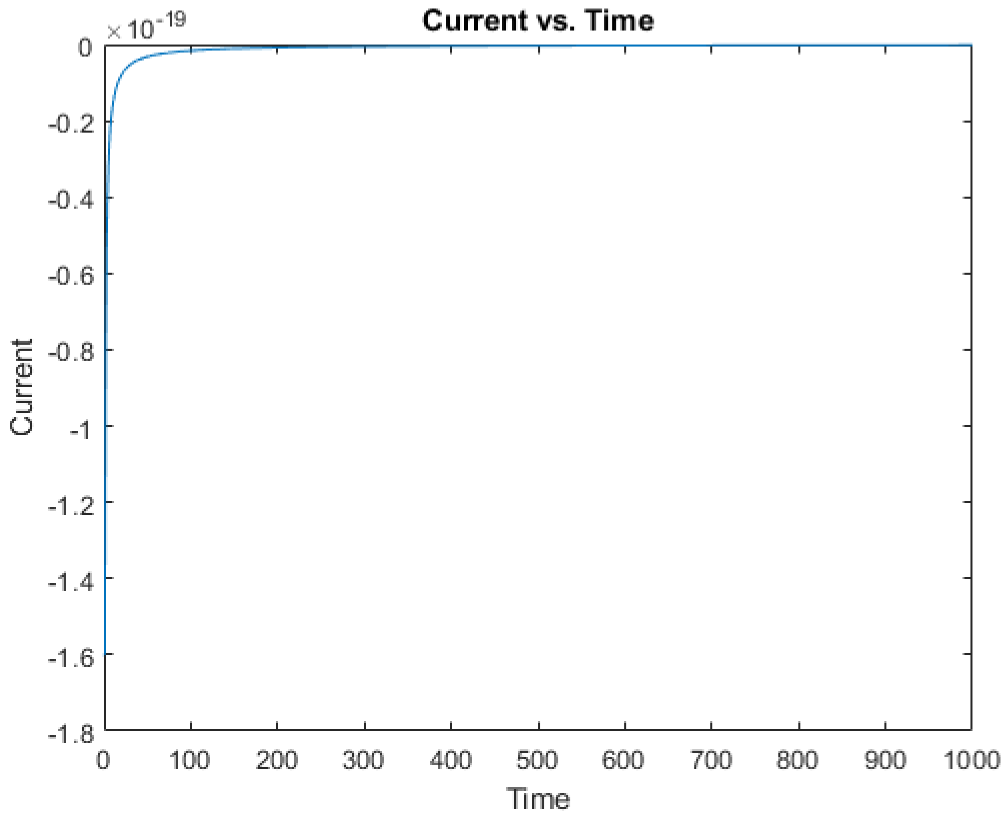


Part 1 D

This section determines how the electron drift current density and average carrier velocity affect each other. A plot of current versus time is shown, along with the formula for current. The formula for electric drift current is J = qn(Un)E.

```
Un = Vx / E;
n = 1000 * (10^15);
J = q * n * (Un) * E;
fprintf('Therefore, the electron current density is %d A/m\n', J);
% From the formula I = q / t, the code below gets current for various
 times
t = 1 : 1000;
I = q ./ t;
figure(2);
plot(t, I);
title('Current vs. Time');
xlabel('Time');
ylabel('Current');
```

```
Therefore, the electron current density is -1.602000e-02 A/m
```

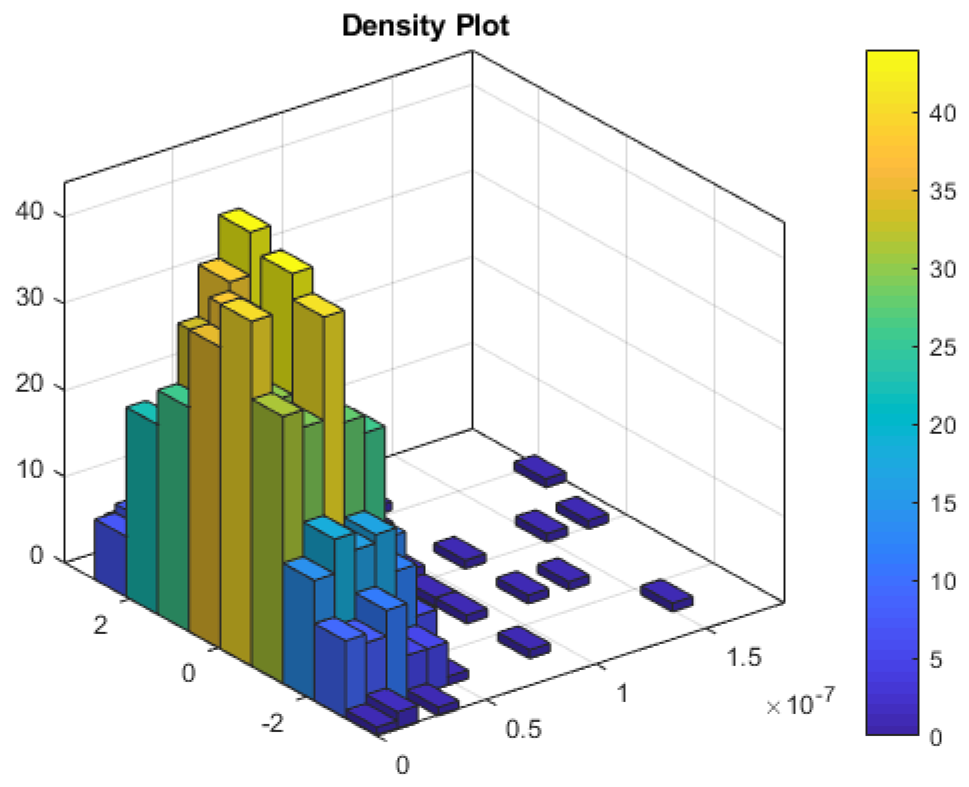


An observation of the figure suggests that the current increases exponentially over time.

Part 1 E

This section produces maps of density and temperature for the semiconductor particles.

```
% Density Plot
figure(3);
histogram2(p(1,:), p(2,:), [20 10], 'FaceColor', 'flat');
colorbar;
title('Density Plot');
```

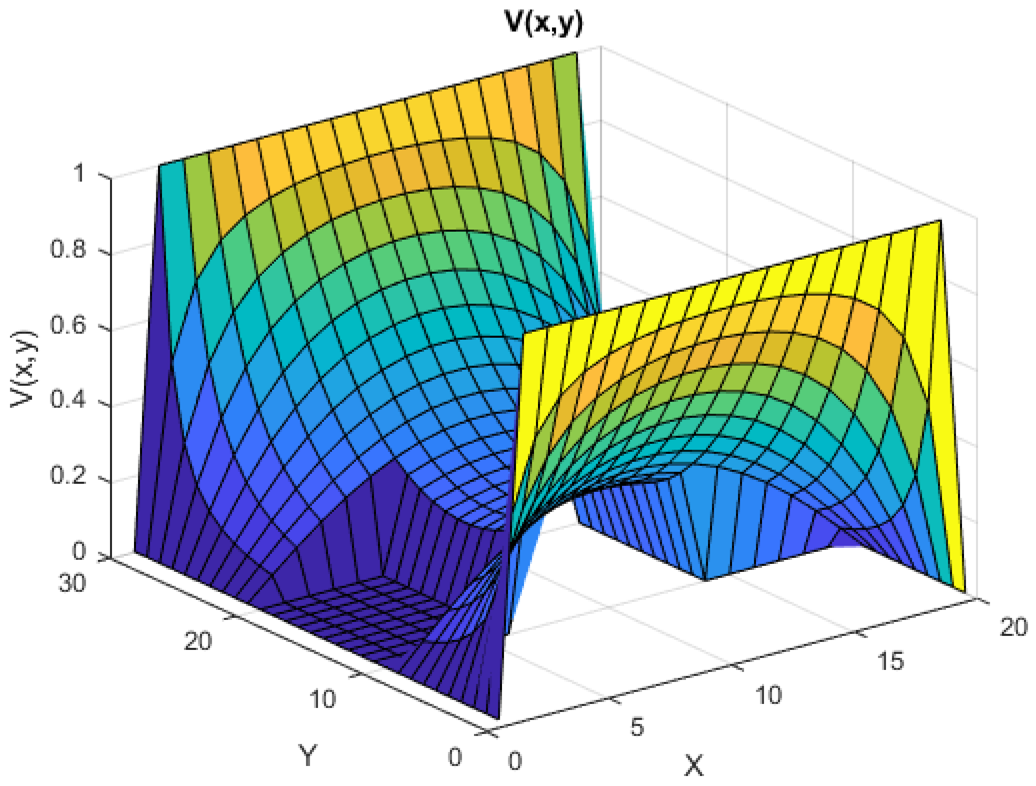


Part 2 A

This section produces a surface plot of the potential with a given 'bottle-neck' region.

```
% Inputs
L = 30;
W = 20;

%Grid
x = linspace(0,L);
y = linspace(0,W);
dx = x(2) - x(1);
dy = y(2) - y(1);

% Make matrices
N = L*W;
M = zeros(N,N);
B = zeros(N,1);

% Interior Points
for i = 2:L-1
    for j = 2:W-1
        n = i + (j-1)*L;
        M(n,n) = -4;
        M(n,n-1) = 1;
        M(n,n+1) = 1;
```

```
            M(n,n-L) = 1;
            M(n,n+L) = 1;
            B(n,1) = 0;
        end
    end

    % Left BC point
    i = 1;
    for j = 1:W
        n = i + (j-1)*L;
        M(n,n) = 1;
         B(n,1) = 1;
    end

    % Right BC
    i = L;
    for j = 1:W
        n = i + (j-1)*L;
        M(n,n) = 1;
        B(n,1) = 0;
    end

    % Bottom BC
    j = 1;
    for i = 1:L
        n = i + (j-1)*L;
        M(n,n) = 1;
    end

    % Top BC
    j = W;
    for i = 1:L
        n = i + (j-1)*L;
        M(n,n) = 1;
    end

    % Solve for potential
    phi_vec = M\B;

    % Convert vector to 2D array
    for i = 1 : L
        for j = 1 : W
            n = i + (j-1)*L;
            phi(i,j) = phi_vec(n);
        end
    end

    % Interior Points
    for i = 2:L-1
        for j = 2:W-1
            n = i + (j-1)*L;
            M(n,n) = -4;
            M(n,n-1) = 1;
            M(n,n+1) = 1;
            M(n,n-L) = 1;
            M(n,n+L) = 1;
            B(n,1) = 0;
        end
    end

    % Left BC point
    i = 1;
    for j = 1:W
        n = i + (j-1)*L;
        M(n,n) = 1;
        B(n,1) = 1;
    end

    % Right BC
    i = L;
    for j = 1:W
        n = i + (j-1)*L;
        M(n,n) = 1;
        B(n,1) = 1;
    end

    % Bottom BC
    j = 1;
    for i = 1:L
        n = i + (j-1)*L;
        M(n,n) = 1;
        B(n,1) = 0;
    end

    % Top BC
    j = W;
    for i = 1:L
        n = i + (j-1)*L;
        M(n,n) = 1;
        B(n,1) = 0;
    end

    % Solve for potential
    phi_vec = M\B;

    % Convert vector to 2D array
    for i = 1 : 30
        for j = 1 : 20
            n = i + (j-1)*L;
            phi(i,j) = phi_vec(n);
        end
    end

    % Sigma
    sigma = ones(L,W);
    for i = 1:L
        for j = 1:W
            if j <= (W/3) || j >= (W*2/3)
                if i >= (L/3) && i <= (L*2/3)
                    sigma(i,j) = 10^-12;
                end

            end
        end
    end

    for i = 1 : 30
        for j = 1 : 20
            if sigma(i,j) == (10^-12)
                phi(i,j) = sigma(i,j);
            end
        end
    end

    % Plot
    figure(5);
    surf(phi);
    xlabel('X');
    ylabel('Y');
    zlabel('V(x,y)');
    title('V(x,y)');
```



Part 2 B

This section determines the electric field, and produces a plot of the electric field vector.

```
% Electric Field
Vmap = zeros(L,W);
for i = 1:L
    for j = 1:W
        n = j + (i-1)*W;
        Vmap(i,j) = phi_vec(n);
    end
end

for i = 1:L
    for j = 1:W
        if i == 1
            Ex(i,j) = (Vmap(i+1,j) - Vmap(i,j));
        elseif i == L
            Ex(i,j) = (Vmap(i,j) - Vmap(i-1,j));
        else
            Ex(i,j) = (Vmap(i+1,j) - Vmap(i-1,j))*0.5;
        end
        if j == 1
            Ey(i,j) = (Vmap(i,j+1) - Vmap(i,j));
        elseif j == W
            Ey(i,j) = (Vmap(i,j) - Vmap(i,j-1));
        else
            Ey(i,j) = (Vmap(i,j+1) - Vmap(i,j-1))*0.5;
        end
    end
end

Ex = -Ex;
Ey = -Ey;
Et = Ex + Ey;
 x1 = 1 : 30;
 y1 = 1 : 20;
[X1, Y1] = meshgrid(x1, y1);
 u = Ex;
 v = Ey;
% figure(6);
% quiver(X1,Y1,Ex,Ey);
```

Part 3 A

This section produces a plot of the electron trajectories under the influence of the new field from Part 2.

```
E = mean(mean(Et));
Vx = E * d;
Mo = 9.109 * (10^-31);
% electron rest mass in kilograms
Mn = 0.26 * Mo;
% effective electron mass in kilograms
```

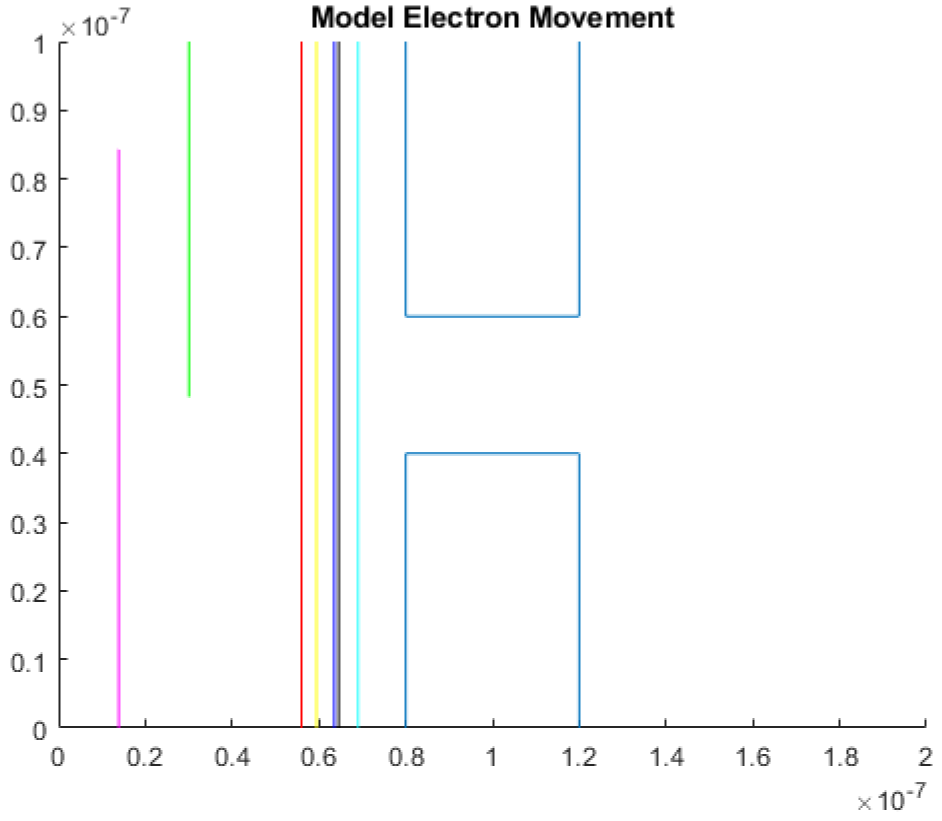
```
L = 200 * (10^-9);
% length of region in metres
W = 100 * (10^-9);
% width of region in metres
T = 300;
% Temperature in Kelvin
Tmn = 0.2 * (10^-12);
% Mean time of collisions between particles in sec
dt = 5 * (10^-15);
% time step in seconds
TStop = 1000 * dt;
% stop simulation after 1000 time steps
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(7);
    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

  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


% This loop updates the positions of the x and y coordinates
for e = 1 : 1 : 1000
     x = rho*(cos(theta(e)));
     y = rho*(sin(theta(e)));
     x = x + (((F/Mn)*dt) * (10^-5));
    rho = sqrt((x^2) + (y^2));
    theta(e) = atan(y/x);
    p(1,e) = p(1,e) + (Vx*(p(1,e)));
    % position gets updated
    if bounce(e) == 1
        p(2,e) = p(2,e) + y;
    else
        p(2,e) = p(2,e) - y;
        % 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
```

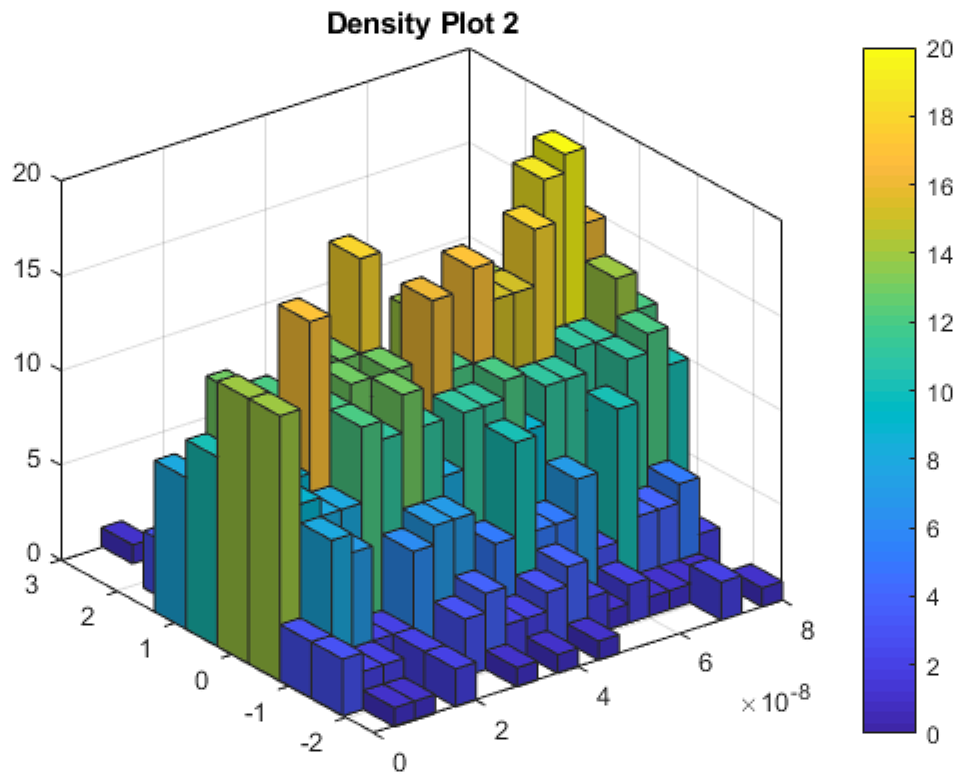
```
        for b = 1 : 1 : 1000
            if p(2,b) < (0.005 * (10^-7))
                % if line hits bottom
                bounce(b) = bounce(b) * (-1);
                % make it go the other way
            end
        end

             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
      end
```



Part 3 B

This section produces a map of the density. Density Plot

```
figure(8);
histogram2(p(1,:), p(2,:), [20 10], 'FaceColor', 'flat');
colorbar;
title('Density Plot 2');
```



Part 3 C

This section determines the next step of improving the accuracy of this simulation of electron trajectories under the influence of electric fields. It would be advantageous to improve the simulation by allowing it to function in 3-D. That is, the plots of the particle movement are currently limited to 2-D, and therefore movement in the x and y dimensions is visible, but is inhibited in the z dimension. A 3-D simulation of particle movement produces a more realistic simulation of how particles behave in the given circumstances. For example, in this simulation the particles can have a curved trajectory, but a 3-D model would be able to show that a particle may have moved in the z-dimension while curving in the x-y plane, and this movement would not be recognizable in the 2-D plot alone.

Published with MATLAB® R2017b