Table of Contents

```
clc
clear
close all
```

ELEC 4700 -- Assignment #3 Submission

Name: Kian Molani, Student Number: 101039806

Assignment Description: In part 01 of this assignment, we will run a Monte Carlo simulation of the trajectories of electrons through an N-type Si semiconductor crystal with potential fields applied across a region of our device. We will also simulate scattering of electrons along their trajectories. In part 02 and 03, we will perform Monte Carlo simulations (w/ scattering) of electron trajectories through a region with some dielectric bottle-neck present. A potential field will be derived from Laplace's equation using Finite Difference techniques, and its effects on electron trajectories will be considered.

Part 01 -- Monte Carlo Simulation of Electron Trajectories w/ Scattering and Applied Potential

The positions of our electrons will be initialized from a uniform distribution, while their velocity components will be sampled from a Maxwell Boltzmann distribution. The standard deviation of a Maxwell Boltzmann distribution for the velocities of electrons in some region at temperature T is given by:

$$\sigma = sqrt(k * T/m_{eff});$$

Of course, a Monte Carlo simulation of electron trajectories entails sampling of electron velocities from this distribution. If we also consider the effects of some constant potential applied across our region, we must derive from this potential the electric field and force acting on our electrons. For a constant voltage applied between two points, the electric field is constant across the region and is given by:

$$E = V/d$$

where V is the voltage applied and d is the distance across our region. To derive the force due to this field acting on some particle with charge q:

$$F = E/q$$

where q is the elementary charge. With a constant electric field applied across our region, the forces acting on each particle are thus all equal. To calculate the accelerations experienced by each each particle due to this force, we can simply make use of Newton's second law (using the effective mass of our electrons through our semiconductor).

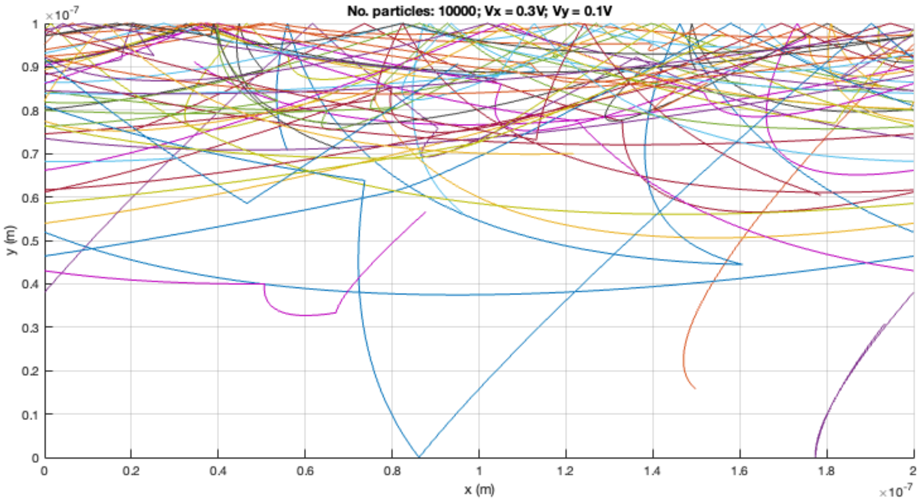
```
% define parameters

CMSQR_TO_MSQR = 0.0001;
e_conc = 1e15 / CMSQR_TO_MSQR; % number of electrons per m^2
l = 200e-9; % region length
w = 100e-9; % region width
area = l * w; % region area
m_0 = 9.10938356e-31; % rest mass of electron
m_eff = 0.26 * m_0; % effective mass of electron through N-type Si
 semiconductor crystal
q = 1.60217662e-19; % electric charge of electron
k = 1.380649e-23; % boltzmann constant
T = 300; % temperature
Vx = 0.3; % voltage across length of semi-conductor region
Vy = 0.1; % voltage across width of semi-conductor region
tao = 0.2e-10; % mean time between electron-electron collisions
num_particles = 10000; % note: must be multiples of 1000
delta_t = 1e-14;
num_iterations = 300;

% calculate number of electrons in region, and how many electrons each
 'particle' represents

num_electrons = e_conc * area;
num_electrons_per_particle = num_electrons / num_particles;

% derive E and F fields and calculate electron accelerations

Ex = Vx / l; Ey = Vy / w;
Fx = q * Ex; Fy = q * Ey;
ax = Fx / m_eff; ay = Fy / m_eff;

% randomly generate positions and sample velocity components from
 Maxwell-Boltzmann distribution

% define mean and standard deviation of distribution
mu = 0;
std = sqrt ( k * T / m_eff );

[x,y,vx,vy,vdx] = initialize_pos(l,w,mu,std,num_particles);

t_since_scatter(1:num_particles) = 0;

% plot electron trajectories w/ scattering and potential fields
 applied

figure(1); set(gcf,'Position',[550 350 800 400]); xlabel('x (m)');
 ylabel('y (m)');
xlim([0 l])
ylim([0 w])
title(strcat("No. particles: ",string(num_particles),"; Vx =
 ",string(Vx),"V; Vy = ",string(Vy),"V"));
grid on
hold on
```

```
colormatrix = [[0, 0.4470, 0.7410];
    [0.8500, 0.3250, 0.0980];
    [0.9290, 0.6940, 0.1250];
    [0.4940, 0.1840, 0.5560];
    [0.4660, 0.6740, 0.1880];
    [0.3010, 0.7450, 0.9330];
    [0.6350, 0.0780, 0.1840];
    [0.75, 0.75, 0];
    [0.25, 0.25, 0.25];
    [0.75, 0, 0.75]];

target_region = [(l - l/10) l]; % target region used to derive current
 density
target_region_area = (target_region(2) - target_region(1)) * w; % area
 of target region expressed in m^2
Jx(1:num_iterations) = 0;
Ix(1:num_iterations) = 0;
time_elapsed(1:num_iterations) = 0;

for i = 1:num_iterations

    % find number of electrons in target region
    n = find(x > target_region(1) & x < target_region(2));

    % calculate current density and current in the x-direction
    if isempty(n)
        Jx(i) = 0;
        Ix(i) = 0;
    else
        Jx(i) = ( q * num_electrons_per_particle * length(n) * vdx ) /
 target_region_area;
        Ix(i) = Jx(i) * target_region_area;
    end
    time_elapsed(i) = delta_t*i;

    x_old(1:num_particles) = x(1:num_particles);
    y_old(1:num_particles) = y(1:num_particles);

    % update positions and velocity components
    x(1:num_particles) = x(1:num_particles) + vx(1:num_particles) *
 delta_t;
    y(1:num_particles) = y(1:num_particles) + vy(1:num_particles) *
 delta_t;
    vx(1:num_particles) = vx(1:num_particles) + ax * delta_t;
    vy(1:num_particles) = vy(1:num_particles) + ay * delta_t;

    % plot x,y positions and check if electrons are at or past
 boundaries
    figure(1);
    for j = 1:num_particles
        if x(j) > l
            m = (y(j)-y_old(j))/(x(j)-x_old(j));
            y_temp = y_old(j)+m*(l-x_old(j));
```

```
           if mod(j,1000) == 0
               plot([x_old(j) l],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = 0 + ( x(j) - l );
           if mod(j,1000) == 0
               plot([0 x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
       elseif x(j) < 0
           m = (y(j)-y_old(j))/(-x(j)+x_old(j));
           y_temp = y_old(j)+m*(x_old(j));
           if mod(j,1000) == 0
               plot([x_old(j) 0],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = l - ( 0 - x(j) );
           if mod(j,1000) == 0
               plot([l x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
       elseif y(j) > w
           m = (y(j)-y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(w-y_old(j))/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
w],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = w - ( y(j) - w );
           if mod(j,1000) == 0
               plot([x_temp x(j)],[w
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       elseif y(j) < 0
           m = (-y(j)+y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(y_old(j))/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
0],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = -y(j);
           if mod(j,1000) == 0
               plot([x_temp x(j)],[0
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       else
           if mod(j,1000) == 0
               plot([x_old(j) x(j)],[y_old(j)
y(j)],'Color',colormatrix(j/1000,1:3));
           end
       end
```

```
        % check to see if electron scatters
        if ( rand() < ( 1 - exp( - t_since_scatter(j) / tao ) ) )

            t_since_scatter(j) = 0;

            % re-sample from Gaussian and reset x,y components of
 velocity vector
            vx(j) = normrnd(mu,std);
            vy(j) = normrnd(mu,std);

        else
            t_since_scatter(j) = t_since_scatter(j) + delta_t;
        end
    end

    % update drift velocity
    vdx = mean(vx);

    pause(0.1);

end
```

```
Warning: Cannot set Position while WindowStyle is 'docked'
```

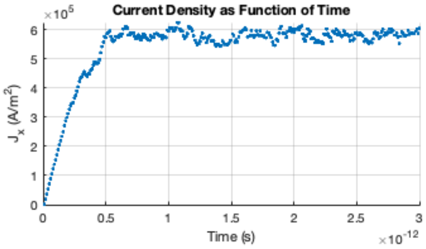


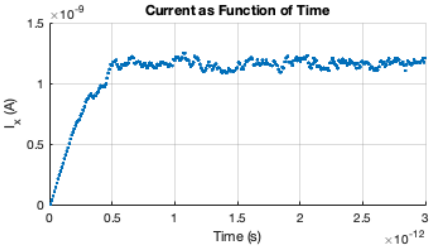
In the code above, current density was calculated using the following formula:

$$J = q * n * v_{dx}$$

where q is elementary charge, n is the number of electrons in our region, and v_{dx} is the average drift velocity of our electrons through our region. Because we are dealing with a 2D simulation, I have decided to manually specify a region one-tenth the size of the entire semi-conductor region (see line 112), and at each timestep counted how many of our simulated particles fall within this region. Given an electron concentration of 10^15 electrons per squared centimeter, and assuming that this concentration can be applied uniformly throughout our device, we can derive how many particles are in our total (L x W) region being analyzed, and divide this number by the number of simulated particles to yield how many electrons each simulated particle represents. For every particle passing through our smaller region used to calculate current density, the variable 'n' will thus be multiplied by this number (i.e. the number of simulated particles found in our region at each timestep will be multiplied by the number of electrons each simulated particle is supposed to represent)

```
% plot current and current density as a function of time

Jx(isnan(Jx))=0; Ix(isnan(Ix))=0;

figure(2)
subplot(2,2,1);
scatter(time_elapsed,Jx,'.');
xlabel('Time (s)'); ylabel('J_x (A/m^2)'); title('Current Density as
 Function of Time');
grid on
subplot(2,2,2);
scatter(time_elapsed,Ix,'.');
xlabel('Time (s)'); ylabel('I_x (A)'); title('Current as Function of
 Time');
grid on
```

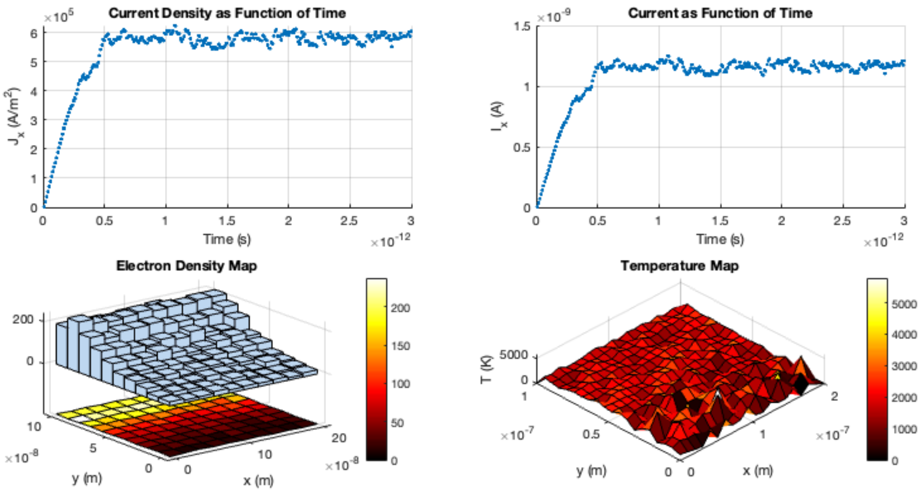




Interestingly, we see our current and current density levelling off to some constant value after a certain period of time has elapsed. This response is exactly what we expect from a system where scattering of electrons are involved. The scattering of our electrons, in other words, prevents them from continuously increasing their velocities without interruption and limits current flow. In fact, if we try setting tao to some high number (say 1e10), our results shows that current continues to increase through the duration of our simulation.

```
% plot density and temperature maps

subplot(2,2,3);
hist3([x',y'])
xlabel('x (m)')
ylabel('y (m)')
hold on
N = hist3([x',y']);
N_pcolor = N';
N_pcolor(size(N_pcolor,1)+1,size(N_pcolor,2)+1) = 0;
```

```
xl = linspace(min(x),max(x),size(N_pcolor,2));
yl = linspace(min(y),max(y),size(N_pcolor,1));
h = pcolor(xl,yl,N_pcolor);
colormap('turbo')
colorbar
h.ZData = -max(N_pcolor(:))*ones(size(N_pcolor));
ax = gca;
ax.ZTick(ax.ZTick < 0) = [];
title('Electron Density Map');

% calculate temperature of each particle
xgrid = linspace(0,l,20);
ygrid = linspace(0,w,20);
T(1:length(xgrid),1:length(ygrid)) = 0;
for i = 1:length(xgrid)-1
    for j = 1:length(ygrid)-1
        count = 1;
        for k = 1:num_particles
            if x(k) < xgrid(i+1) && x(k) > xgrid(i) && y(k) < ygrid(j
+1) && y(k) > ygrid(j)
                v = sqrt ( vx(k)^2 + vy(k)^2 );
                kinetic_energy = 1/2 * m_eff * v^2;
                T(i,j) = T(i,j) + kinetic_energy / 1.380649e-23;
                count = count + 1;
            end
        end
        T(i,j) = T(i,j) / count;
    end
end

subplot(2,2,4);
surf(xgrid',ygrid',T')
colormap('hot')
view(315,75);
colorbar; title('Temperature Map');
xlabel('x (m)'); ylabel('y (m)'); zlabel('T (K)');
```



Of course, with a voltage applied along the width of our semi-conductor region, and since we've simulated our electrons to bounce back off our top and bottom walls, electrons are going to be concentrated along either the top or bottom of our device. To yield temperature maps, we've grouped our region into smaller bins, and averaged the temperature of our particles in each of those bins using standard kinetic energy formulas relating temperature to velocity. Because of the constant E-field applied across our device, our electrons don't tend to travel faster in one part of a region than another, which yields a temperature map that looks more or less uniform.

Part 02 -- MC Simulation of Electron Trajectories through a Bottleneck, w/ Potential Field Calculated using Finite Difference

As stated previously, in this part of the assignment, electron trajectories are simulate through a region with a dielectric bottleneck present. Our potential field is not constant this time, and was derived from Laplace's equation using finite difference techniques. From the results of this computation, we can derive the force applied and accelerations experienced on/by each electron the exact same way we did for part 01. For a more thorough explanation of this code's functionality, please refer to my lab report for assignment #2.

```
clc
clear
close all

% define parameters

m_0 = 9.10938356e-31; % rest mass of electron
m_eff = 0.26 * m_0; % effective mass of electron through N-type Si
 semiconductor crystal
k = 1.380649e-23; % boltzmann constant
T = 300; % temperature
q = 1.60217662e-19; % electric charge of electron
l = 200e-9; % region length
w = 100e-9; % region width
nx = 100; ny = 100;
L_b = 0.2 * nx;
W_b = 0.4 * ny;
sigma_inside = 1e-2; % conductivity inside boxes
sigma_outside = 1; % conductivity outside boxes
bottle_neck_width = 1; % default value (listed as a factor)
tao = 1e-3; % mean time between electron-electron collisions
num_particles = 10000; % note: must be multiples of 1000
delta_t = 0.3e-5;
num_iterations = 1000;

box_x_1 = nx/2-(L_b/2); box_x_2 = nx/2+(L_b/2);
box_y_1 = W_b; box_y_2 = ny-W_b;

% compute results for V and E

[~,Vmap,Ex,Ey,~,~,~] =
 compute_params(nx,ny,L_b,W_b,sigma_inside,sigma_outside,bottle_neck_width);
```

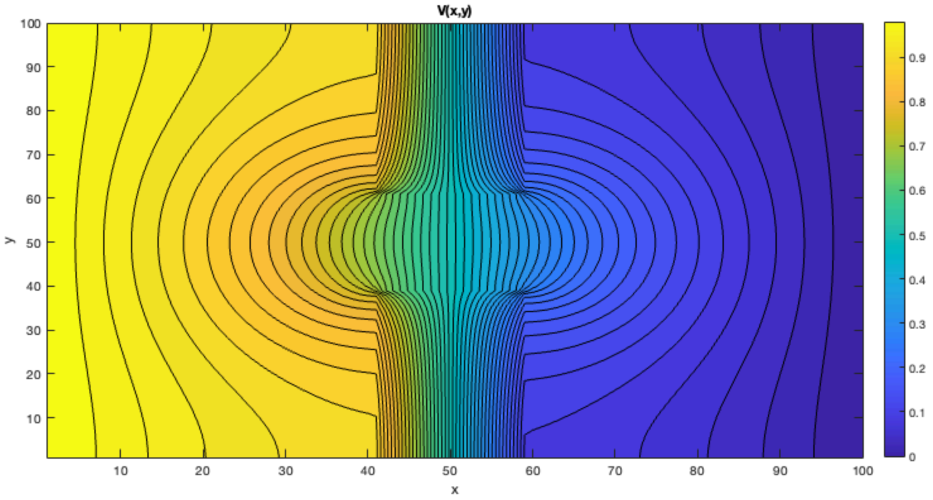
```
% plot V and E fields

figure(3);
contourf(Vmap,50)
colorbar
xlabel('x'); ylabel('y'); title('V(x,y)');

figure(4);
quiver(Ex,Ey);
xlim([0,nx]); ylim([0,ny]); xlabel('x'); ylabel('y'); title('E(x,y)');

% derive F field and calculate electron accelerations

Fx = q * Ex; Fy = q * Ey;
ax = Fx / m_eff; ay = Fy / m_eff;

% randomly generate positions and sample velocity components from
 Maxwell-Boltzmann distribution

% define mean and standard deviation of distribution
mu = 0;
std = sqrt ( k * T / m_eff );

% randomly sample speeds from Gaussian distribution
vx = normrnd(mu,std,1,num_particles);
vy = normrnd(mu,std,1,num_particles);

% define drift velocity in x-direction as the mean of our velocities
 in x-direction
vdx = mean(vx);

% randomly sample positions from uniform distribution and ensure
 particles don't spawn in boxes
i = 1;
while i <= num_particles
    tempx = rand(1) * nx;
    tempy = rand(1) * ny;

    if ( tempx < nx/2-(L_b/2) || tempx > nx/2+(L_b/2) )
        x(i) = tempx;
        y(i) = tempy;
        i = i +1;
    elseif tempy > W_b && tempy < ny-W_b
        x(i) = tempx;
        y(i) = tempy;
        i = i + 1;
    end
end

t_since_scatter(1:num_particles) = 0;

% plot electron trajectories w/ scattering and potential fields
 applied
```

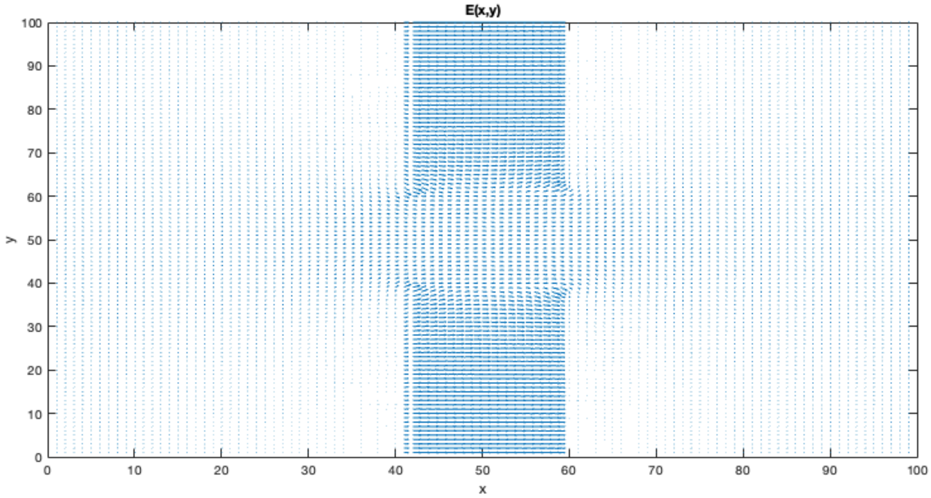
```
figure(5); xlabel('x'); ylabel('y');
rectangle('Position',[nx/2-L_b/2 0 L_b W_b]);
rectangle('Position',[nx/2-L_b/2 ny-W_b L_b W_b]);
xlim([0 nx])
ylim([0 ny])
title(strcat("No. particles: ",string(num_particles)));
grid on
hold on

colormatrix = [[0, 0.4470, 0.7410];
    [0.8500, 0.3250, 0.0980];
    [0.9290, 0.6940, 0.1250];
    [0.4940, 0.1840, 0.5560];
    [0.4660, 0.6740, 0.1880];
    [0.3010, 0.7450, 0.9330];
    [0.6350, 0.0780, 0.1840];
    [0.75, 0.75, 0];
    [0.25, 0.25, 0.25];
    [0.75, 0, 0.75]];

for i = 1:num_iterations

    x_old(1:num_particles) = x(1:num_particles);
    y_old(1:num_particles) = y(1:num_particles);

    % plot x,y positions and check if electrons are at or past
 boundaries
    figure(5);
    for j = 1:num_particles
        % update position and velocity components
        x(j) = x(j) + vx(j) * delta_t;
        y(j) = y(j) + vy(j) * delta_t;

        rndx = round(x(j)); rndy = round(y(j));
        if rndx == 0
            rndx = 1;
        elseif rndx < 0
            rndx = 1;
        elseif rndx > 100
            rndx = 100;
        end
        if rndy == 0
            rndy = 1;
        elseif rndy < 0
            rndy = 1;
        elseif rndy > 100
            rndy = 100;
        end

        vx(j) = vx(j) + ax(rndx,rndy) * delta_t;
        vy(j) = vy(j) + ay(rndx,rndy) * delta_t;

        if x(j) > nx
            m = (y(j)-y_old(j))/(x(j)-x_old(j));
```

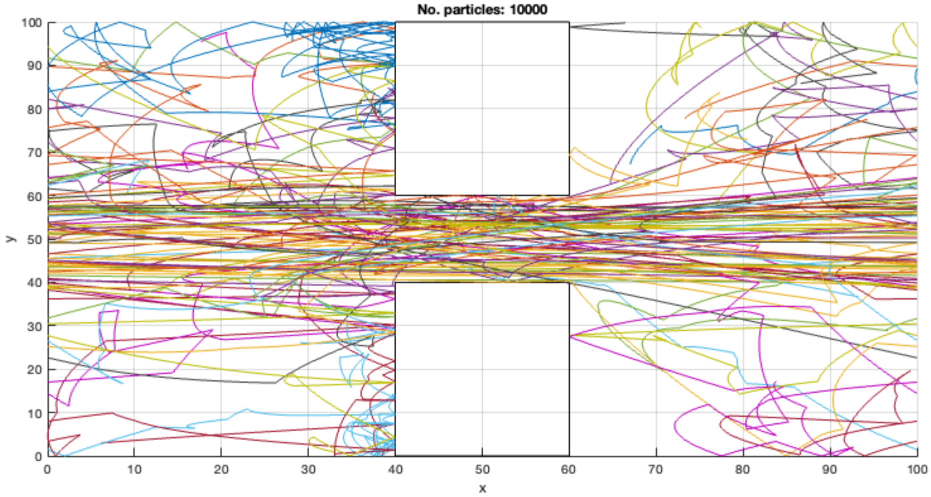
```
           y_temp = y_old(j)+m*(nx-x_old(j));
           if mod(j,1000) == 0
               plot([x_old(j) nx],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = 0 + ( x(j) - nx );
           if mod(j,1000) == 0
               plot([0 x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
       elseif x(j) < 0
           m = (y(j)-y_old(j))/(-x(j)+x_old(j));
           y_temp = y_old(j)+m*(x_old(j));
           if mod(j,1000) == 0
               plot([x_old(j) 0],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = nx - ( 0 - x(j) );
           if mod(j,1000) == 0
               plot([nx x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
       elseif y(j) > ny
           m = (y(j)-y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(ny-y_old(j))/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
ny],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = ny - ( y(j) - ny );
           if mod(j,1000) == 0
               plot([x_temp x(j)],[ny
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       elseif y(j) < 0
           m = (-y(j)+y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(y_old(j))/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
0],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = -y(j);
           if mod(j,1000) == 0
               plot([x_temp x(j)],[0
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       elseif x(j) > box_x_1 && x(j) < box_x_2 && ( y(j) < box_y_1 ||
y(j) > box_y_2) && x_old(j) < box_x_1
           m = (y(j)-y_old(j))/(x(j)-x_old(j));
           y_temp = y_old(j)+m*(box_x_1-x_old(j));
           if mod(j,1000) == 0
```

```
               plot([x_old(j) box_x_1],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = box_x_1 - ( x(j) - box_x_1 );
           if mod(j,1000) == 0
               plot([box_x_1 x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vx(j) = -vx(j);
       elseif x(j) > box_x_1 && x(j) < box_x_2 && ( y(j) < box_y_1 ||
y(j) > box_y_2) && x_old(j) > box_x_2
           m = (y(j)-y_old(j))/(-x(j)+x_old(j));
           y_temp = y_old(j)+m*(x_old(j)-box_x_2);
           if mod(j,1000) == 0
               plot([x_old(j) box_x_2],[y_old(j)
y_temp],'Color',colormatrix(j/1000,1:3));
           end
           x(j) = box_x_2 + ( box_x_2 - x(j) );
           if mod(j,1000) == 0
               plot([box_x_2 x(j)],[y_temp
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vx(j) = -vx(j);
       elseif x(j) > box_x_1 && x(j) < box_x_2 && y(j) < box_y_1
           m = (-y(j)+y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(y_old(j)-box_y_1)/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
box_y_1],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = box_y_1 + ( box_y_1 - y(j) );
           if mod(j,1000) == 0
               plot([x_temp x(j)],[box_y_1
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       elseif x(j) > box_x_1 && x(j) < box_x_2 && y(j) > box_y_2
           m = (y(j)-y_old(j))/(x(j)-x_old(j));
           x_temp = x_old(j)+(box_y_2-y_old(j))/m;
           if mod(j,1000) == 0
               plot([x_old(j) x_temp],[y_old(j)
box_y_2],'Color',colormatrix(j/1000,1:3));
           end
           y(j) = box_y_2 - ( y(j) - box_y_2 );
           if mod(j,1000) == 0
               plot([x_temp x(j)],[box_y_2
y(j)],'Color',colormatrix(j/1000,1:3));
           end
           vy(j) = -vy(j);
       else
           if mod(j,1000) == 0
               plot([x_old(j) x(j)],[y_old(j)
y(j)],'Color',colormatrix(j/1000,1:3));
           end
```

```
        end

        % check to see if electron scatters
        if ( rand() < ( 1 - exp( - t_since_scatter(j) / tao ) ) )

            t_since_scatter(j) = 0;

            % re-sample from Gaussian and reset x,y components of
 velocity vector
            vx(j) = normrnd(mu,std);
            vy(j) = normrnd(mu,std);

        else
            t_since_scatter(j) = t_since_scatter(j) + delta_t;
        end
    end

    pause(0.1);

end
```



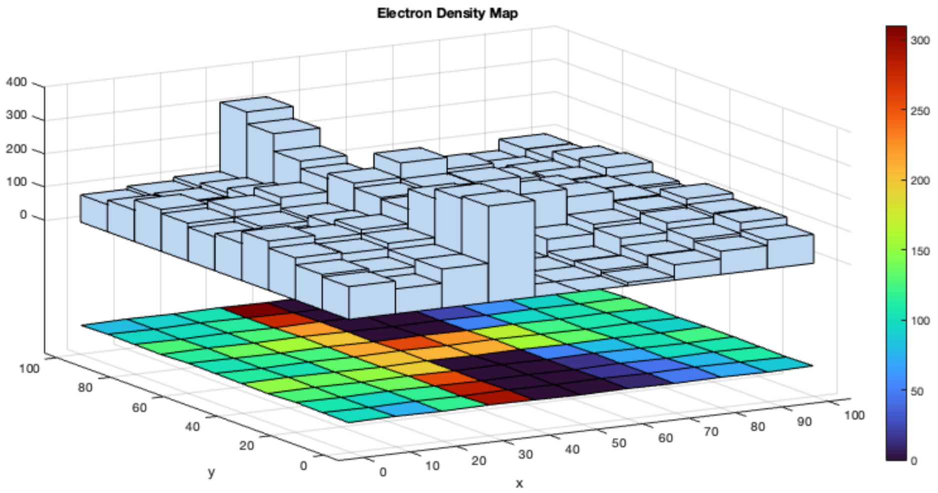


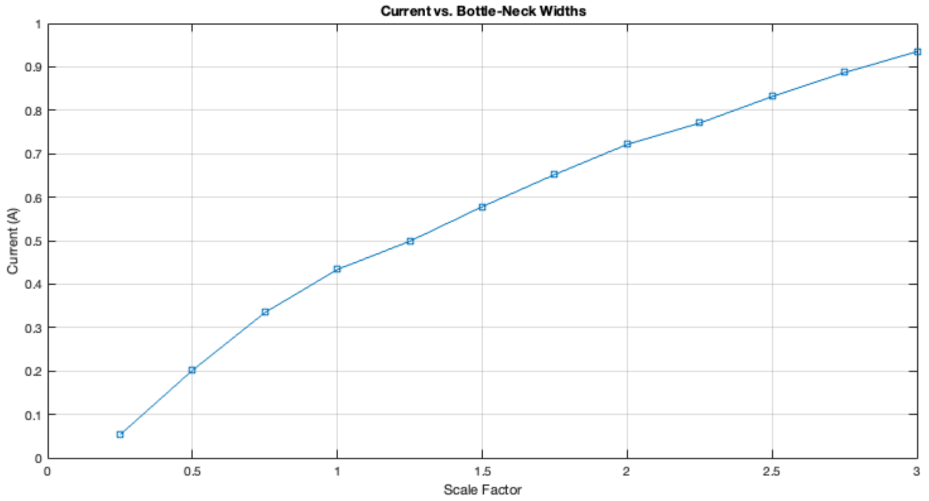


Part 03 -- Extracting Additional Parameters from Bottle-Neck Simulation

```
% plot density map

figure(6);
hist3([x',y'])
xlabel('x')
ylabel('y')
hold on
N = hist3([x',y']);
N_pcolor = N';
N_pcolor(size(N_pcolor,1)+1,size(N_pcolor,2)+1) = 0;
xl = linspace(min(x),max(x),size(N_pcolor,2));
yl = linspace(min(y),max(y),size(N_pcolor,1));
h = pcolor(xl,yl,N_pcolor);
colormap('turbo')
```

```
colorbar
h.ZData = -max(N_pcolor(:))*ones(size(N_pcolor));
ax = gca;
ax.ZTick(ax.ZTick < 0) = [];
title('Electron Density Map');
view(330,25);

% compute and plot results for variations in bottle-neck width

factors = 0; factors = [0.25:0.25:3];
current_matrix = 0; current_matrix(1:length(factors)) = 0;

for i=1:length(factors)
    [sigma_matrix,Vmap,Ex,Ey,Jx,Jy,I] =
 compute_params(nx,ny,L_b,W_b,sigma_inside,sigma_outside,factors(i));
    current_matrix(i) = I;
end

figure;
plot(factors,current_matrix,'-s');
xlabel('Scale Factor');
ylabel('Current (A)');
title('Current vs. Bottle-Neck Widths');
grid on
```





Our density map illustrates that because our potential field is applied left-to-right, the bottle-neck acts as a barrier to impede the flow of most electrons. As such, we can see a relative accumulation of electrons on the left-hand side of our barrier. For electrons to pass the barrier, they must go through our bottle. Because of this, and because of the relatively high E-field in that region, we also see a high concentration of electrons and current flow through our bottle-neck.

In general, to improve the results of our simulation, we can do a few things. In my implementation, I divided my region into a 100x100 points and calculated the E-field at each point using the finite difference method. The force acting on each particle was thus set according to the E-field closest to (but not exactly at) its position. Thus, by increasing the number of bins/points used for finite difference, we can increase the accuracy of our simulation results. Of course, this has the disadvantage of taking more time to simulate and/or requiring more computational resources. As with any simulation, we can also improve our results by better simulating the underlying physics involved (this might mean invoking phenomenon apart from just scattering to better account for the trajectories of electrons. For smaller regions, for instance, we might decide to account for effects on current due to quantum tunneling). As an example, we might decide to use an alternate expression to yield better results for the probability of scattering at each timestep (one that takes into account the density of material and the distance travelled between successive collisions).

Published with MATLAB® R2020b