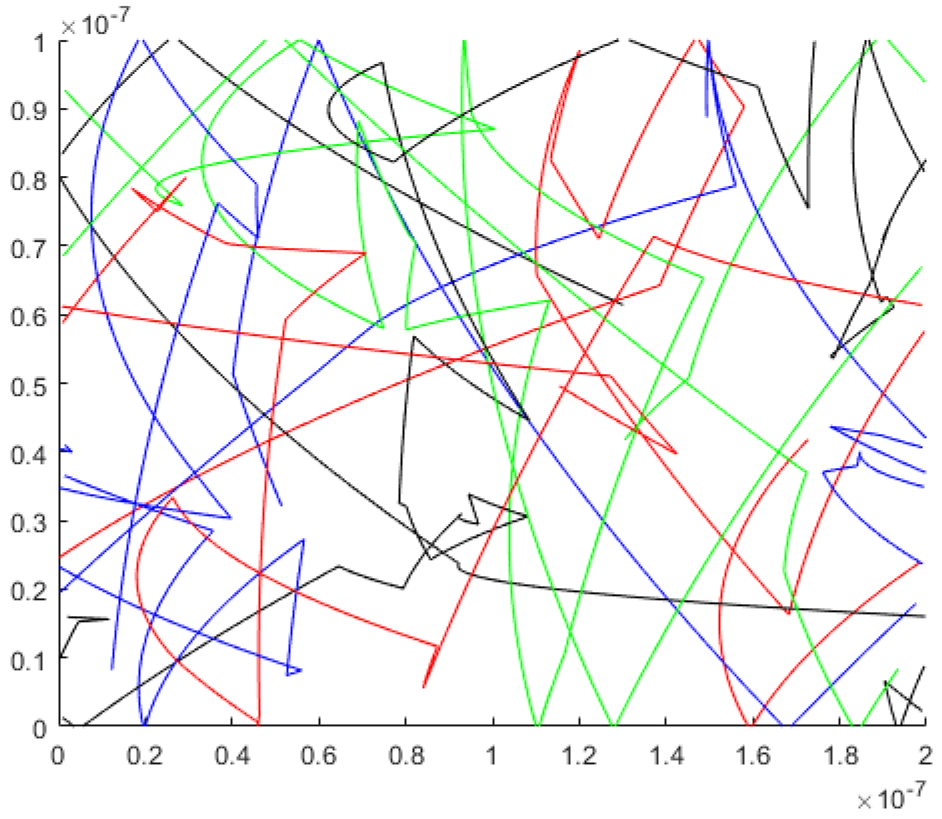
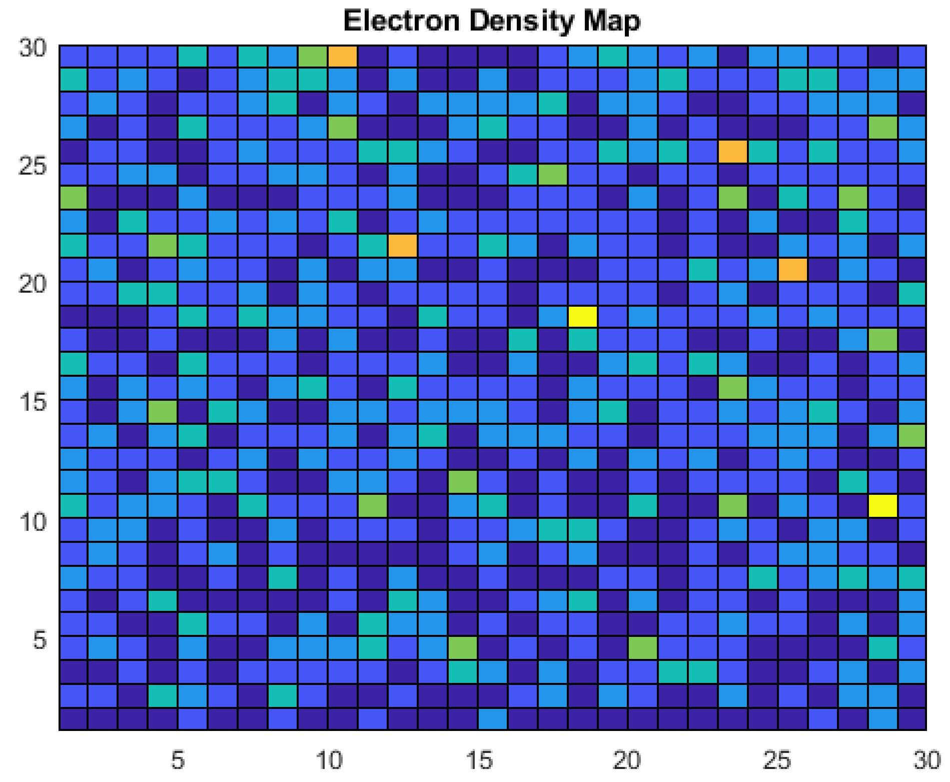
Section 1: Constant Electric Field

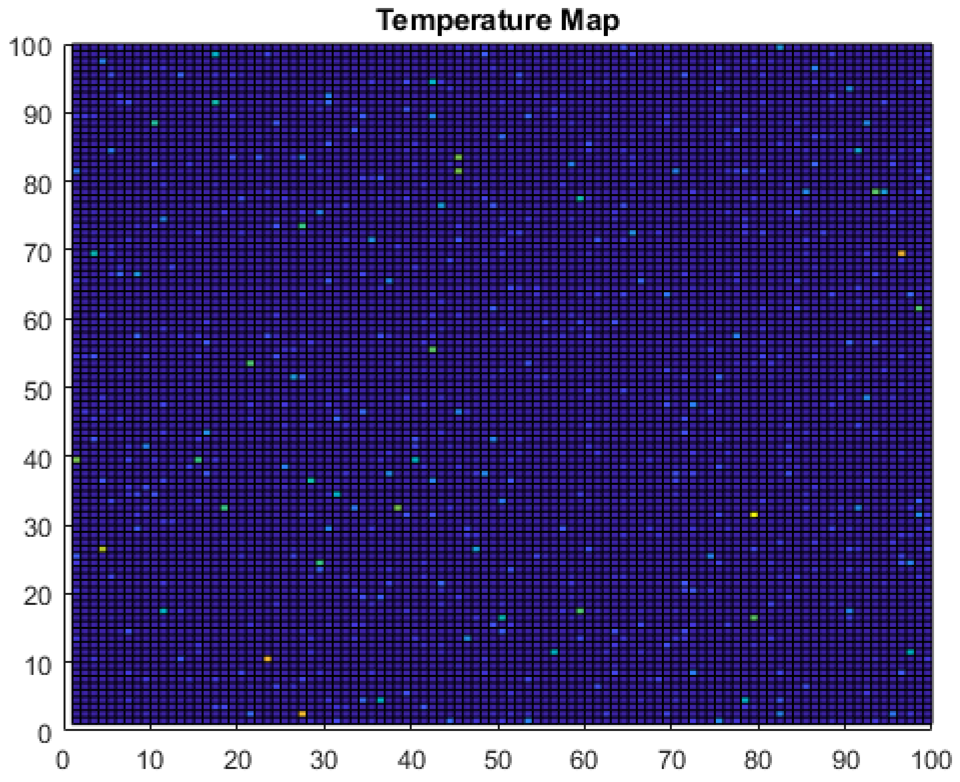
In this section, a constant electric field is applied in the X-direction of the simulation plane. Each electron experiences an equal force from the field, since the magnitude of the field is equal everywhere, and the electric force is F = eE, which is of course also constant for all involved electrons in the system. The acceleration for each particle is then A = eE/m, which is also constant for every particle. Using a value of 1V shows a very defined curving to the right, which leaves an electric field of 1e7V/m. The magnitude of the force experienced by each particle is then 1.602e-12 N, which in turn leads to an acceleration of 1.7588m/s^2. The current density component from electron drift is given by J = qnuE, where u is the mobility of electrons, E is the electric field, n is the number of free electrons, and q is the elementary charge.

```
clear
C.q_0 = 1.60217653e-19;
C.m_0 = 9.10938215e-31;
C.kb = 1.3806504e-23;
C.T = 300;
frameWidth = 200e-9;
frameHeight = 100e-9;
nAtoms = 1000;
Vth = sqrt(2*C.kb*C.T /(0.26*C.m_0));
dt = frameHeight/Vth/100;
Tstop = 750*dt;
t = 0;
freepath = 0.2e-12;
Pscatter = 1 - exp(-dt/freepath);
Voltage = 1;
Efield = Voltage / frameWidth;
Force = Efield * C.q_0;
Accel = Force / C.m_0;
J = C.q_0*nAtoms*Efield;

%initializing vectors
Xnext = zeros(1,nAtoms);
Ynext = zeros(1,nAtoms);
VX = Vth * randn(1,nAtoms);
VY = Vth * randn(1,nAtoms);
V = sqrt(VY.*VY+VX.*VX);
X = frameWidth * rand(1, nAtoms);
Y = frameHeight * rand(1, nAtoms);
R = zeros(1, nAtoms);
Temperature = zeros(1, 100);
iteration = 1;


while t < Tstop
    %determines which particles scatter and performs calculations on
 them
    %to determine mean free path and time between collisions
    R = rand(1,nAtoms);
    VX(R<Pscatter) = Vth*randn(1);
    VY(R<Pscatter) = Vth*randn(1);
    VX = VX + Accel*dt;
    V = sqrt(VY.*VY+VX.*VX);


    Xnext = X + VX*dt;
    Ynext = Y + VY*dt;
    %X boundary conditions set
    right = Xnext>frameWidth;
    left = Xnext<0;
    Xnext(right) = Xnext(right)-frameWidth;
    Xnext(left) = Xnext(left) + frameWidth;
    %Y boundary conditions set
    top = Ynext > frameHeight;
    bottom = Ynext < 0;
    VY(top | bottom) = VY(top | bottom) * -1;
    %calculations for temperature
    Temperature(iteration) = 0.26*C.m_0*mean(V.^2)/4/C.kb;
    figure(1)
    xlim([0 frameWidth])
    ylim([0 frameHeight])
    hold on
    %plotting, but avoid plotting the full horizontal jump
    if abs(Xnext(1) - X(1)) < 2*abs(VX(1))*dt
        figure(1)
        plot([Xnext(1) X(1)], [Ynext(1) Y(1)], 'blue')
    end
    if abs(Xnext(2) - X(2)) < 2*abs(VX(2))*dt
        figure(1)
        plot([Xnext(2) X(2)], [Ynext(2) Y(2)], 'red')
    end
    if abs(Xnext(3) - X(3)) < 2*abs(VX(3))*dt
        figure(1)
        plot([Xnext(3) X(3)], [Ynext(3) Y(3)], 'green')
    end
    if abs(Xnext(4) - X(4)) < 2*abs(VX(4))*dt
        figure(1)
        plot([Xnext(4) X(4)], [Ynext(4) Y(4)], 'black')
    end

    %updating positions, and advancing time a step forward so the
 while
    %loop works
    X = Xnext;
    Y = Ynext;
    t = t+dt;
    iteration = iteration + 1;
    pause(0.0001);
end


%electron density map
figure(2)
EDM = hist3([X',Y'],[30,30]);
pcolor(EDM')
title('Electron Density Map')
```

```
view(2)

%temperature map
xLim = linspace(0,frameWidth,100);
yLim = linspace(0,frameHeight,100);
xTempReg = discretize(X,xLim);
yTempReg = discretize(Y,yLim);
for q=1:1:100
    for w=1:1:100
        %Temperature contained in defined region
        tempReg = (q == xTempReg) & (w == yTempReg);

        %Total velocities in region
        vxTot=sum(VX(tempReg));
        vyTot=sum(VY(tempReg));
        vTot = sqrt((vxTot)^2+(vyTot)^2);

        %Calculate Temperature
        tempMap(q,w) = C.m_0*0.26*(vTot)^2/(2*C.kb);
    end
end
figure(3)
surf(tempMap)
view(2)
title('Temperature Map')
```




Electron Density Map
30
25
20
15
10
5
5
10
15
20
25
30

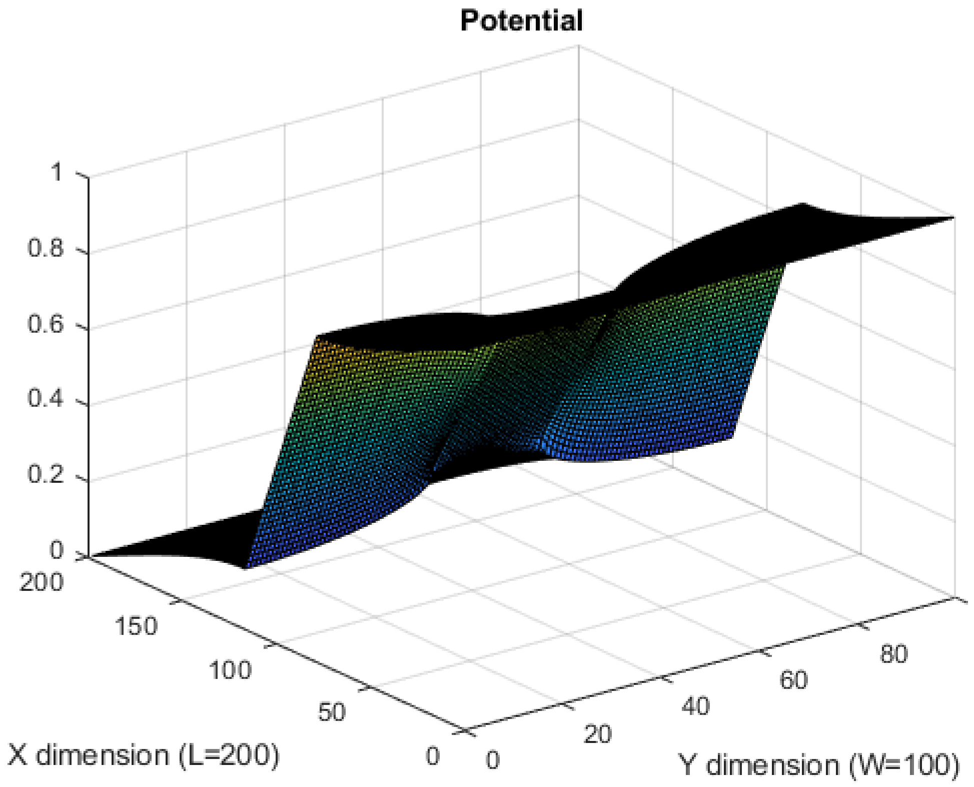

Temperature Map
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100

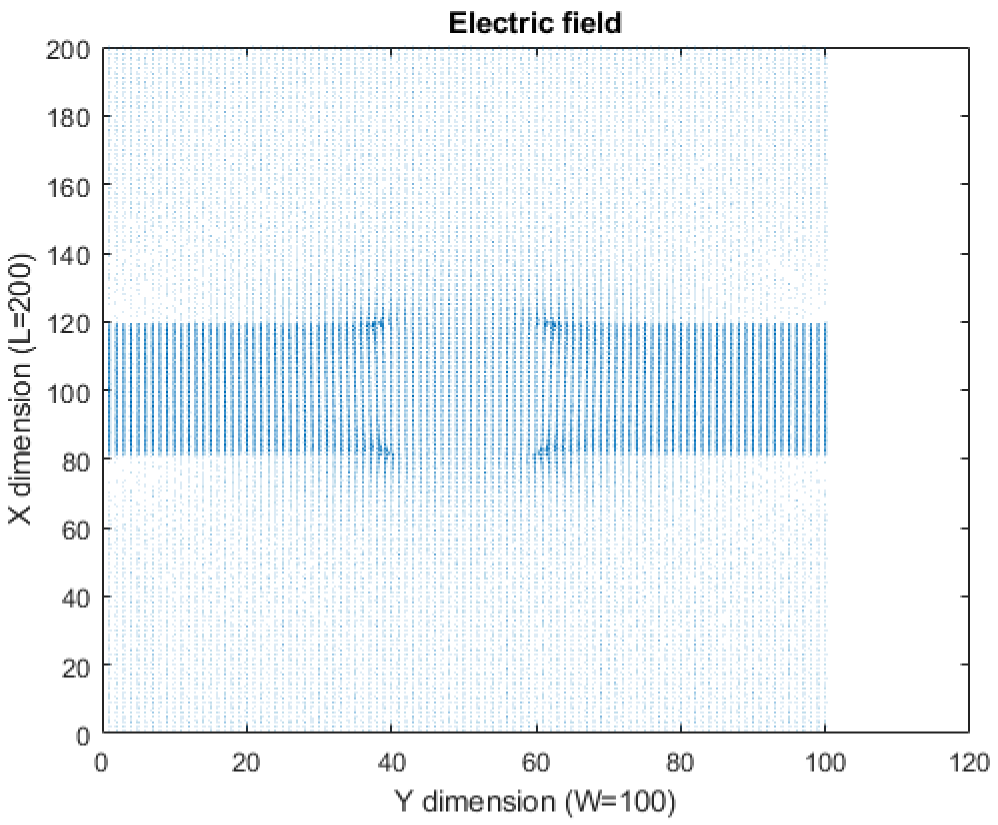
Finite Difference Solution of Electric Field

This section prepares the electric field solution for use in the next section of the assignment. In the previous section, we assumed a constant field across the entirety of the simulation window. Here, we have a non-constant conductivity across the field of view. The different boundary conditions produce different results for the potential and electric field, both of which are plotted. The finite difference matrix method is used, in exactly the same way as in assignment 2.

```
nx = 200;
ny = 100;
G = sparse(nx*ny, nx*ny);
B = zeros(1,nx*ny);
%setting conductivity map for 3rd part
cMap = ones(nx,ny);
 for q = 1:nx
     for w = 1:ny
         if ((q<(0.6*nx)&&q>(0.4*nx)&&w>(0.6*ny)) ||
 (q<(0.6*nx)&&q>(0.4*nx)&&w<(0.4*ny)))
             cMap(q,w) = 1e-2;
         end
     end
 end

for i = 1:nx
    for j = 1:ny
        %Setting up the G-Matrix
        n = j + (i-1)*ny;
        if i == 1
            G(n,n) = 1;
            B(n) = 1;
        elseif i == nx
            G(n,n) = 1;
        elseif j == 1
            nxm = j + (i-2)*ny;
            nxp = j + (i)*ny;
            nyp = j+1 + (i-1)*ny;

            rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
            rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
            ryp = (cMap(i,j) + cMap(i,j+1))/2.0;

            G(n,n) = -(rxm + rxp + ryp);
            G(n,nxm) = rxm;
            G(n,nxp) = rxp;
            G(n,nyp) = ryp;
        elseif j == ny
            nxm = j + (i-2)*ny;
            nxp = j + (i)*ny;
            nym = j-1 + (i-1)*ny;

            rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
            rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
```

```
                rym = (cMap(i,j) + cMap(i,j-1))/2.0;

                G(n,n) = -(rxm + rxp + rym);
                G(n,nxm) = rxm;
                G(n,nxp) = rxp;
                G(n,nym) = rym;
            else
                nxm = j + (i-2)*ny;
                nxp = j + (i)*ny;
                nym = j-1 + (i-1)*ny;
                nyp = j+1 + (i-1)*ny;

                rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
                rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
                rym = (cMap(i,j) + cMap(i,j-1))/2.0;
                ryp = (cMap(i,j) + cMap(i,j+1))/2.0;

                G(n,nxp) = rxp;
                G(n,nyp) = ryp;
                G(n,nxm) = rxm;
                G(n,nym) = rym;
                G(n,n) = -(rxm + rxp + rym + ryp);
            end
        end
    end
    %solving for matrix of potentials
    V = G\B';
    Vmap = zeros(nx,ny);
    for i = 1:nx
        for j = 1:ny
            n = j + (i-1)*ny;
            Vmap(i,j) = V(n);
        end
    end


    %plotting potential
    figure(4)
    surf(Vmap)

    title('Potential')
    ylabel('X dimension (L=200)')
    xlabel('Y dimension (W=100)')

    %plotting electric field
    [Ex,Ey] = gradient(-Vmap);
    figure(5)
    quiver(Ex,Ey)
    title('Electric field')
    ylabel('X dimension (L=200)')
    xlabel('Y dimension (W=100)')
```


Potential
1
0.8
0.6
0.4
0.2
0
200
150
100
50
0
X dimension (L=200)
0
20
40
60
80
100
Y dimension (W=100)


Electric field
200
180
160
140
120
100
80
60
40
20
0
X dimension (L=200)
0
20
40
60
80
100
120
Y dimension (W=100)

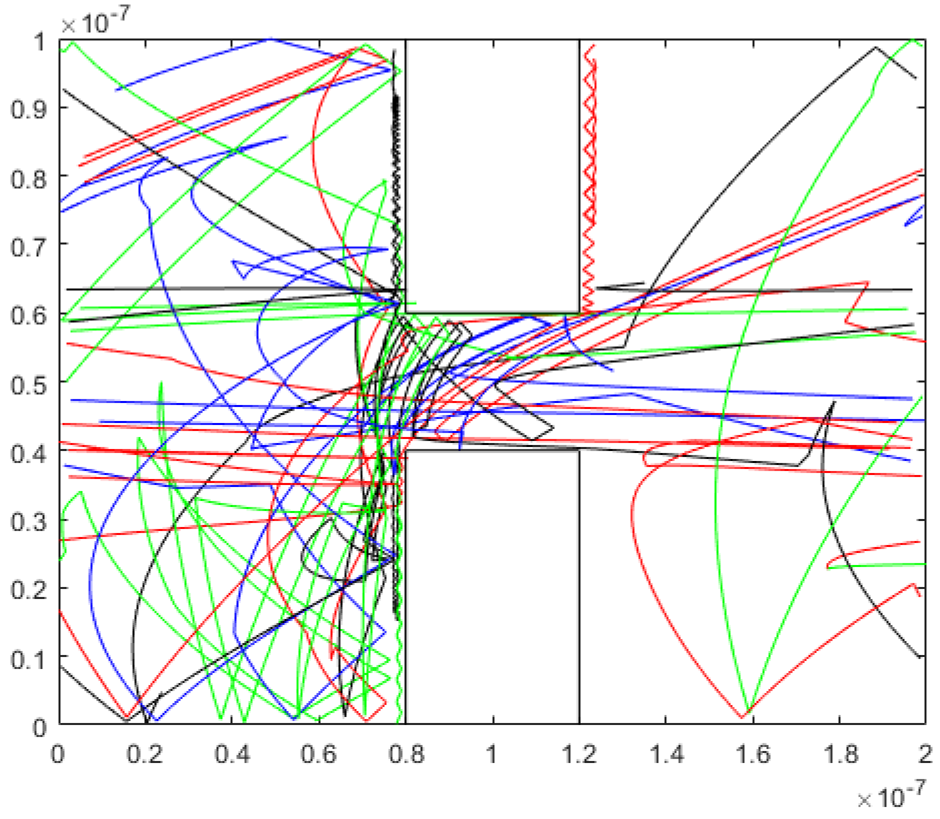
Section 3 - Coupled Simulations

In this section of the assignment, I have coupled the electron trajectory simulations with the finite difference solution of the electric fields. The electrons respond to the electric field by having curved trajectories, pointing along the direction of the field, or towards the right hand side of the simulation window. specular reflection is used for the boundaries of the bottleneck, and the top and bottom boundaries of the simulation window. The solution of the electric field is not changed at all from the previous section, and is only done once, as opposed to doing it every loop. The particles still exhibit their scattering behaviour, to keep the conservation of energy intact for the system. A cool trick using the sub2ind function is used to eliminate the need for a nested for loop inside the main while loop to update the velocities of the particles based on what cell of the window they reside in. This trick speeds up the simulation significantly compared to the nested loop method. To enhance this simulation further, it could become significantly more complicated by recognizing that the current flow through the system would generate magnetic fields, which would in turn change the electric field present in the system. A super complicated Yee cell method could be employed, but it would be unnecessary and tragically difficult for me to implement it with my current Matlab skillset.

```
nx = 200;
ny = 100;
G = sparse(nx*ny, nx*ny);
B = zeros(1,nx*ny);
%setting conductivity map for 3rd part
cMap = ones(nx,ny);
 for q = 1:nx
     for w = 1:ny
         if ((q<(0.6*nx)&&q>(0.4*nx)&&w>(0.6*ny)) ||
 (q<(0.6*nx)&&q>(0.4*nx)&&w<(0.4*ny)))
             cMap(q,w) = 1e-2;
         end
     end
 end

for i = 1:nx
    for j = 1:ny
        %Setting up the G-Matrix
        n = j + (i-1)*ny;
        if i == 1
            G(n,n) = 1;
            B(n) = 3;
        elseif i == nx
            G(n,n) = 1;
        elseif j == 1
            nxm = j + (i-2)*ny;
            nxp = j + (i)*ny;
            nyp = j+1 + (i-1)*ny;

            rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
            rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
            ryp = (cMap(i,j) + cMap(i,j+1))/2.0;

            G(n,n) = -(rxm + rxp + ryp);
            G(n,nxm) = rxm;
            G(n,nxp) = rxp;
```

```
                G(n,nyp) = ryp;
            elseif j == ny
                nxm = j + (i-2)*ny;
                nxp = j + (i)*ny;
                nym = j-1 + (i-1)*ny;

                rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
                rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
                rym = (cMap(i,j) + cMap(i,j-1))/2.0;

                G(n,n) = -(rxm + rxp + rym);
                G(n,nxm) = rxm;
                G(n,nxp) = rxp;
                G(n,nym) = rym;
            else
                nxm = j + (i-2)*ny;
                nxp = j + (i)*ny;
                nym = j-1 + (i-1)*ny;
                nyp = j+1 + (i-1)*ny;

                rxm = (cMap(i,j) + cMap(i-1,j))/2.0;
                rxp = (cMap(i,j) + cMap(i+1,j))/2.0;
                rym = (cMap(i,j) + cMap(i,j-1))/2.0;
                ryp = (cMap(i,j) + cMap(i,j+1))/2.0;

                G(n,nxp) = rxp;
                G(n,nyp) = ryp;
                G(n,nxm) = rxm;
                G(n,nym) = rym;
                G(n,n) = -(rxm + rxp + rym + ryp);
            end
        end
    end
    %solving for matrix of potentials
    V = G\B';
    Vmap = zeros(nx,ny);
    for i = 1:nx
        for j = 1:ny
            n = j + (i-1)*ny;
            Vmap(i,j) = V(n);
        end
    end
    [Ex,Ey] = gradient(-Vmap);
    %from here nothing above really matters, the field is solved and its
     all
    %good


    C.q_0 = 1.60217653e-19;
    C.m_0 = 9.10938215e-31;
    C.kb = 1.3806504e-23;
    C.T = 300;
    frameWidth = 200e-9;
    frameHeight = 100e-9;
```

```
nAtoms = 10;
Vth = sqrt(2*C.kb*C.T /(0.26*C.m_0));
dt = frameHeight/Vth/100;
Tstop = 750*dt;
t = 0;
freepath = 0.2e-12;
Pscatter = 1 - exp(-dt/freepath);
AccelX = 2e9* Ey .* C.q_0 ./ C.m_0;
AccelY = 1e9* Ex .* C.q_0 ./ C.m_0;

%initializing vectors
Xnext = zeros(1,nAtoms);
Ynext = zeros(1,nAtoms);
VX = Vth * randn(1,nAtoms);
VY = Vth * randn(1,nAtoms);
V = sqrt(VY.*VY+VX.*VX);
X = frameWidth * rand(1, nAtoms);
Y = frameHeight * rand(1, nAtoms);
R = zeros(1, nAtoms);
Temperature = zeros(1, 100);
iteration = 1;

%defining box boundaries
boxleft = X>0.8e-7;
boxright = X<1.2e-7;
box1bottom = Y>0.6e-7;
box2top = Y<0.4e-7;
boxedin = (boxleft&boxright&box1bottom)|(boxleft&boxright&box2top);
%removing particles from the boxes
while sum(boxedin)>0
X(boxedin) = rand*frameWidth;
Y(boxedin) = rand*frameHeight;
boxleft = X>0.8e-7;
boxright = X<1.2e-7;
box1bottom = Y>0.6e-7;
box2top = Y<0.4e-7;
boxedin = (boxleft&boxright&box1bottom)|(boxleft&boxright&box2top);
end
%all particles are out of the boxes to begin with

while t < Tstop

    Xindex = ceil(X * 1e9);
    Yindex = ceil(Y * 1e9);
    %scatter
    R = rand(1,nAtoms);
    VX(R<Pscatter) = Vth*randn(1);
    VY(R<Pscatter) = Vth*randn(1);
    %accelerate
    ain = sub2ind(size(AccelX),Xindex, Yindex);
    aiin = sub2ind(size(AccelY),Xindex, Yindex);
    VX = VX + AccelX(ain)*dt;
    VY = VY + AccelY(aiin)*dt;
    V = sqrt(VY.*VY+VX.*VX);
    %set next
    Xnext = X + VX*dt;
    Ynext = Y + VY*dt;
    %X boundary conditions set
    right = Xnext>frameWidth;
    left = Xnext<0;
    Xnext(right) = Xnext(right)-frameWidth;
    Xnext(left) = Xnext(left) + frameWidth;
    %Y boundary conditions set
    top = Ynext > frameHeight;
    bottom = Ynext < 0;
    VY(top | bottom) = VY(top | bottom) * -1;

    %set boundary for sides of boxes
    boxsides = (Ynext > 0.6e-7 | Ynext <
0.4e-7)&(Xnext>0.75e-7)&(Xnext<1.25e-7);
    topbox =
(Y<0.60e-7)&(Ynext>0.60e-7)&(Xnext>0.8e-7)&(Xnext<1.2e-7);
    bottombox =
(Y>0.40e-7)&(Ynext<0.40e-7)&(Xnext>0.8e-7)&(Xnext<1.2e-7);
    VY(topbox|bottombox) = VY(topbox|bottombox) * -1;
    VX(boxsides) = VX(boxsides) * -1;

    Ynext = Y + VY*dt;
    %calculations for temperature
    Temperature(iteration) = 0.26*C.m_0*mean(V.^2)/4/C.kb;
    figure(6)
    xlim([0 frameWidth])
    ylim([0 frameHeight])
    plot([0.8e-7 0.8e-7],[0 0.4e-7], 'black')
    plot([1.2e-7 1.2e-7],[0 0.4e-7], 'black')
    plot([0.8e-7 1.2e-7],[0.4e-7 0.4e-7], 'black')
    plot([0.8e-7 0.8e-7],[1e-7 0.6e-7], 'black')
    plot([1.2e-7 1.2e-7],[1e-7 0.6e-7], 'black')
    plot([0.8e-7 1.2e-7],[0.6e-7 0.6e-7], 'black')
    hold on
    %plotting, but avoid plotting the full horizontal jump
    if abs(Xnext(1) - X(1)) < 2*abs(VX(1))*dt
        figure(6)
        plot([Xnext(1) X(1)], [Ynext(1) Y(1)], 'blue')
    end
    if abs(Xnext(2) - X(2)) < 2*abs(VX(2))*dt
        figure(6)
        plot([Xnext(2) X(2)], [Ynext(2) Y(2)], 'red')
    end
    if abs(Xnext(3) - X(3)) < 2*abs(VX(3))*dt
        figure(6)
        plot([Xnext(3) X(3)], [Ynext(3) Y(3)], 'green')
    end
    if abs(Xnext(4) - X(4)) < 2*abs(VX(4))*dt
        figure(6)
        plot([Xnext(4) X(4)], [Ynext(4) Y(4)], 'black')
    end
    %updating positions, and advancing time a step forward so the
 while
    %loop works
    X = Xnext;
    Y = Ynext;
    t = t+dt;
    iteration = iteration + 1;
    pause(0.0001);
end
```



Published with MATLAB® R2018a