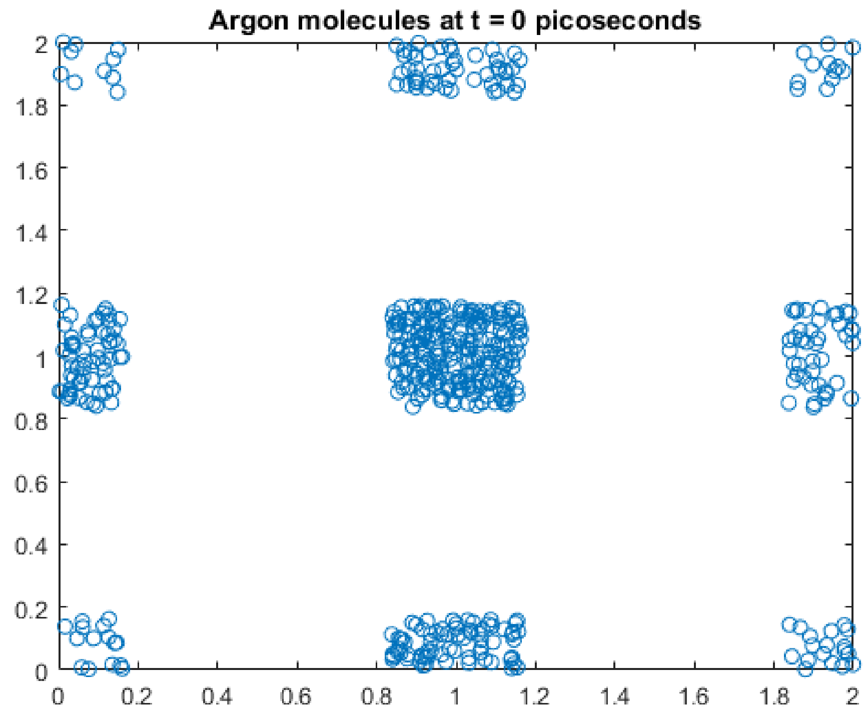
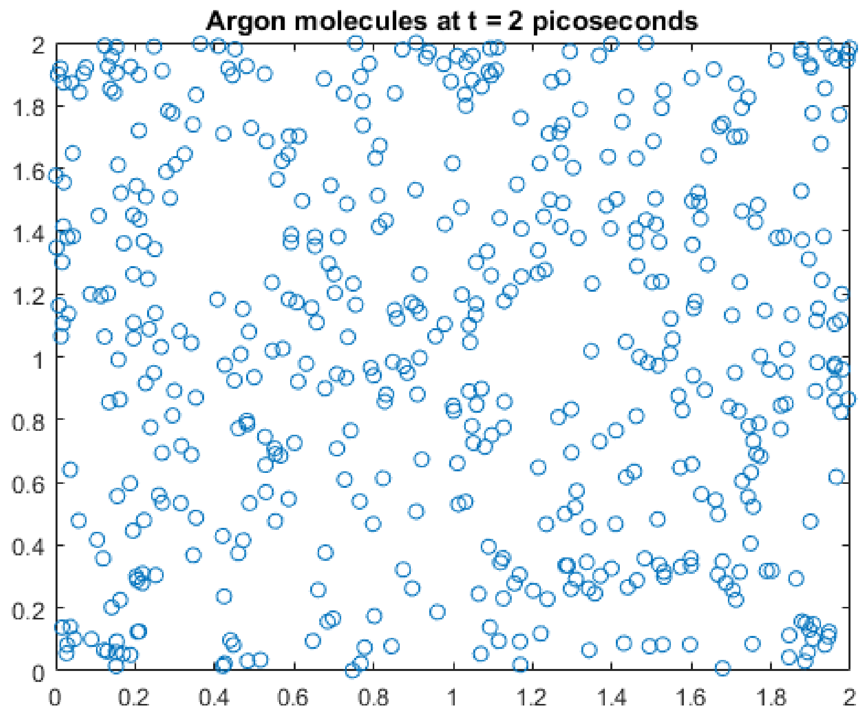
Christopher Wong
999234204
MAT 128C

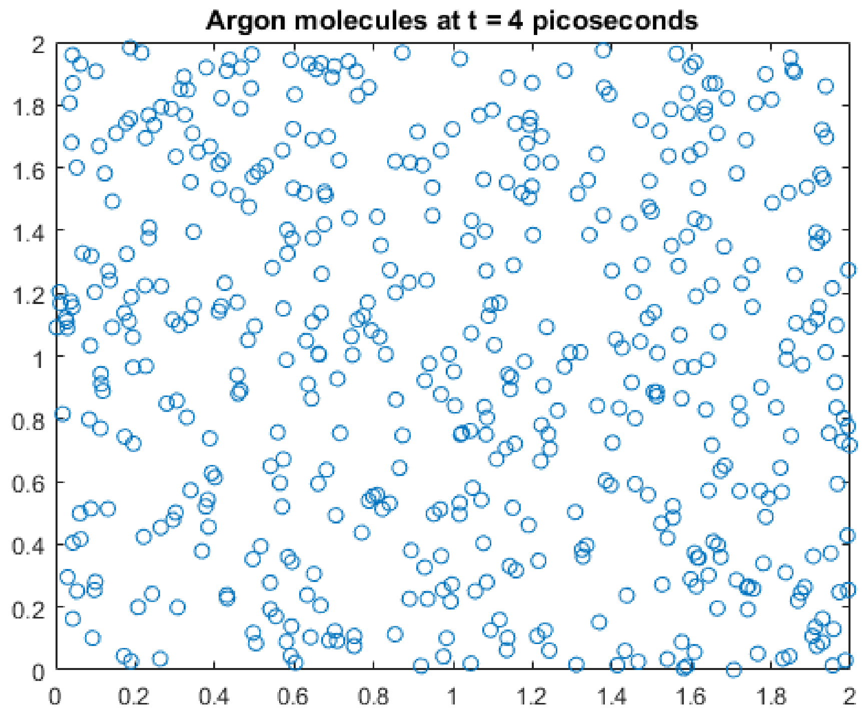
Project 2

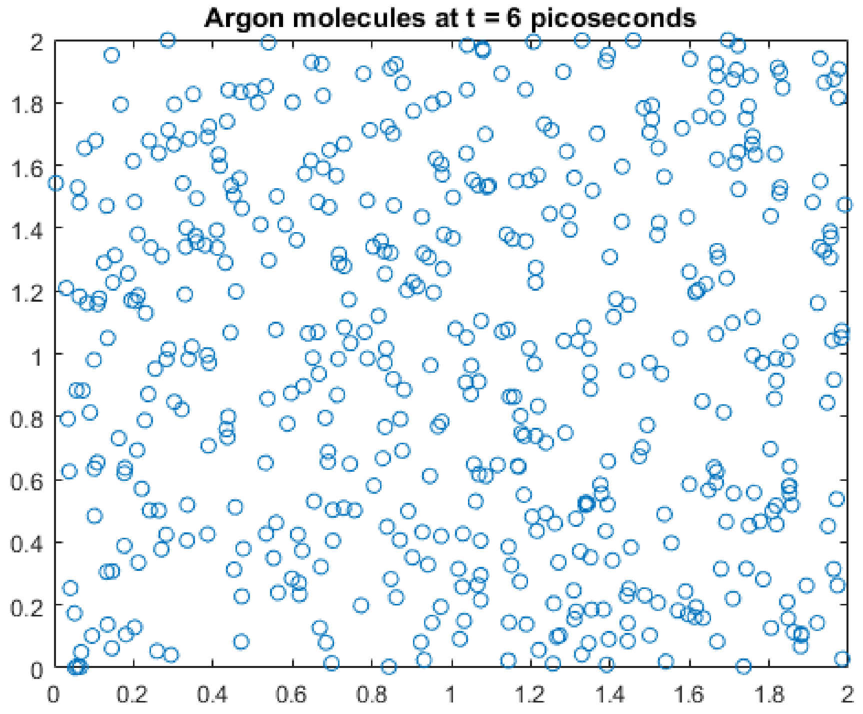
Project_2.m takes in
N = Number of atoms in the system
L = Size of the system (boundaries of $[0, 2\sigma L]$)
T = Number of time steps

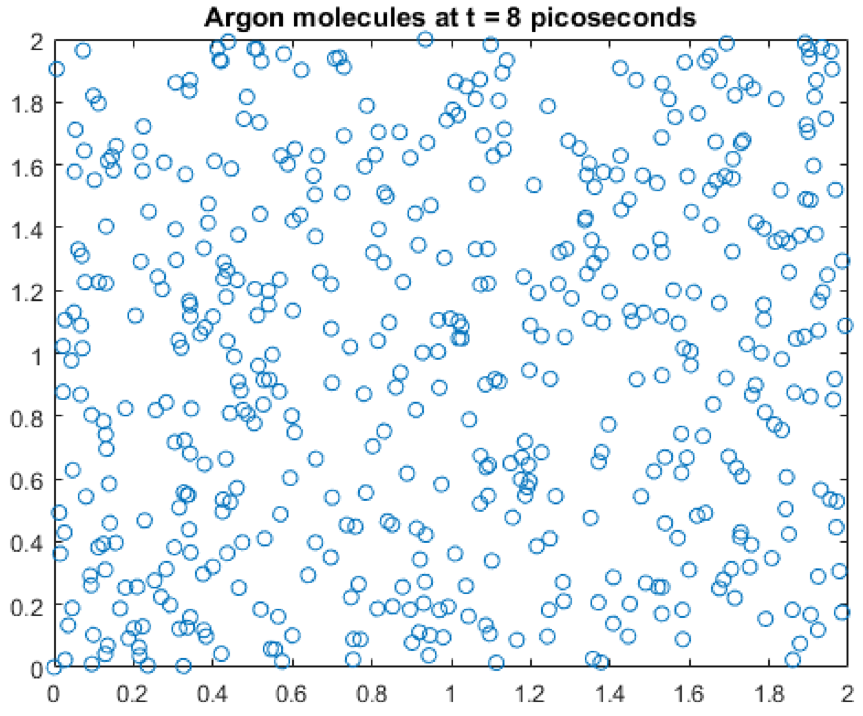
with $\Delta T = 2$ picoseconds and $\sigma = 1$. It then calculates and stores the positions of each Argon atom at each time step in a matrix x and y for the x and y coordinates of the atom, respectively. Then it plots the results. We set the seed with $rng(123)$ and simulate 500 atoms in a $([0, 2] \times [0, 2])$ system. The first 10 picoseconds are given below:

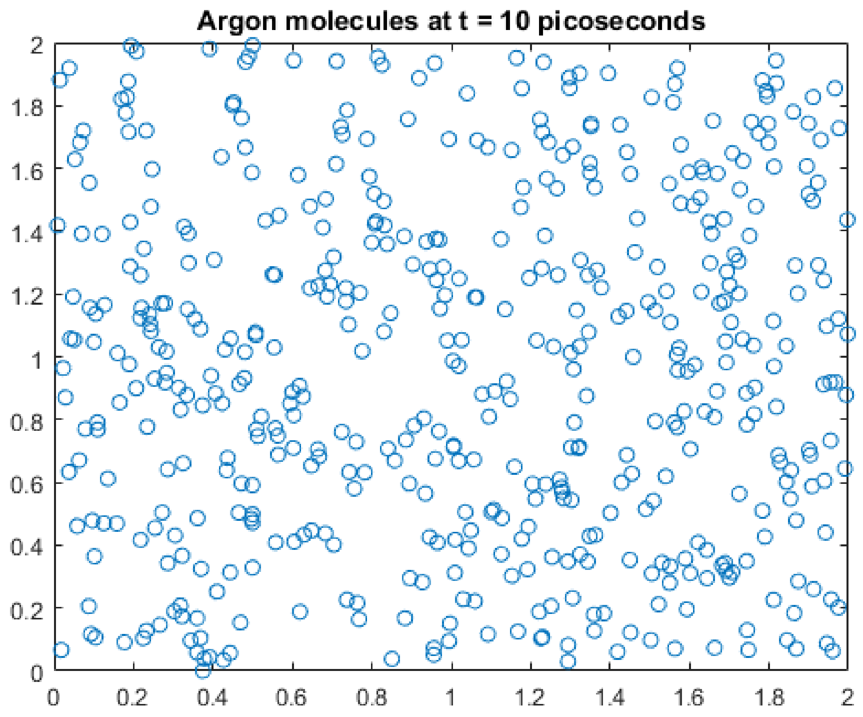



Argon molecules at t = 2 picoseconds
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Argon molecules at t = 4 picoseconds
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Argon molecules at t = 6 picoseconds
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Argon molecules at t = 8 picoseconds
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2



I am not sure if my calculated temperatures are correct because the velocities go from $v_0 = 0.1$ to $v_1 > 10^9$ as a result of dividing by $2\Delta T = (4 * 10^{-12}$ seconds) and thus the temperatures end up as:

Time (ps)	Temperature (K)
0	7.655274297457165e-03
2	7.410371621844270e+50
4	6.837365137519428e+42
6	1.306611796774299e+47
8	5.967677920824267e+57
10	3.250203995660685e+50

Project_2.m

```
%%% MAT 128C Project 2

format long e
clear all
% cd 'C:\Users\Christopher Wong\Desktop\MAT 128C\Project 2';

% rng(123)
% Initialization of the system
N = input('Number of atoms: ');
L = input('Size of system: ');
T = input('Number of time steps: ');
% 2 picoseconds = 10^-12 seconds
dt = 2*10^-12;
% Reduced units
sigma = 1;

% Boundary conditions
[gridX, gridY] = meshgrid(0:sigma:2*L*sigma);

% Assign initial positions
posX = randi(size(gridX-1, 1), N, 1);
posY = randi(size(gridY-1, 1), N, 1);

% Displace the N points
delta_r = (sigma/2).*rand(1);
posX = mod(posX + (2.*(rand(N, 1)-0.5)).*delta_r, 2*L*sigma);
posY = mod(posY + (2.*(rand(N, 1)-0.5)).*delta_r, 2*L*sigma);

% Assign initial velocities
v0x = repelem(0.1,numel(posX))';
v0y = repelem(0.1,numel(posY))';
velX(:,1) = 2*(rand(N, 1)-0.5).*v0x;
velY(:,1) = 2*(rand(N, 1)-0.5).*v0y;

%%% Verlet algorithm
% x and y directions
% Column 1 corresponds to t = -1
x(:,1) = mod(posX - velX(:,1).*dt, 2*sigma*L);
y(:,1) = mod(posY - velY(:,1).*dt, 2*sigma*L);
x(:,2) = mod(posX, 2*sigma*L);
y(:,2) = mod(posY, 2*sigma*L);
for n = 2:(T+1)
    atoms = [x(:,n) y(:,n)];
    for j = 1:N
        for i = 1:j
            % Calculate distances between atoms (across boundaries)
            if (atoms(j,1) < atoms(i,1))
                xdist = atoms(j,1) + (2*sigma*L) - atoms(i,1);
                deltaX(j,i) = min((atoms(i,1) - atoms(j,1)), xdist);
            else
                xdist = atoms(i,1) + (2*sigma*L) - atoms(j,1);
                deltaX(j,i) = min((atoms(j,1) - atoms(i,1)), xdist);
            end
            if (atoms(j,2) < atoms(i,2))
                ydist = atoms(j,2) + (2*sigma*L) - atoms(i,2);
                deltaY(j,i) = min((atoms(i,2) - atoms(j,2)), ydist);
            else
                ydist = atoms(i,2) + (2*sigma*L) - atoms(j,2);
                deltaY(j,i) = min((atoms(j,2) - atoms(i,2)), ydist);
            end
            boundary_dist(i,j) = sqrt(xdist^2 + ydist^2);
        end
    end
    deltaX = deltaX' + deltaX;
    deltaY = deltaY' + deltaY;
    r = min(squareform(pdist(atoms)), boundary_dist);
    r = r' + r;
    for j = 1:N
        for i = 1:j
            % Derivative of energy given by Van der Walls interactions
            % Reduce computation time
            if (r(i,j) > 3*sigma)
                f(i,j) = 0;
            end
        end
    end
    f = 24.*((2./(r.^13)) - 1./(r.^7));
    f = f' + f;
    % Cosines used in calculating acceleration
    cosineX = deltaX ./ r;
    cosineY = deltaY ./ r;
    for i = 1:N
        accelX(i,n) = nansum(f(i,:).*cosineX(i,:));
        accelY(i,n) = nansum(f(i,:).*cosineY(i,:));
    end
    x(:,n+1) = 2.*x(:,n) - x(:,n-1) + accelX(:,n).*dt^2;
    y(:,n+1) = 2.*y(:,n) - y(:,n-1) + accelY(:,n).*dt^2;
    % Calculating velocity components
    % Column 1 corresponds to t = 0
    velX(:,n) = (x(:,n+1) - x(:,n-1)) / (2*dt);
    velY(:,n) = (y(:,n+1) - y(:,n-1)) / (2*dt);
    x(:,n+1) = mod(x(:,n+1), 2*sigma*L);
    y(:,n+1) = mod(y(:,n+1), 2*sigma*L);
end
```

```
% Comparing data against known results
% Mass of 1 Argon molecule = 6.69 x 10^(-26) kg
m = 6.69*10^(-26);
sys_mass = N*m;
% Boltzmann constant 1.38064852  10-23 m2 kg s-2 K-1
K_B = 1.38064852 * 10^(-23);
vel(1) = mean(velX(:,1).^2 + velY(:,1).^2);
temperature(1) = (sys_mass/(2*K_B))*(vel(1));
for i = 2:(size(velX,2))
    vel(i) = mean(velX(:,i).^2 + velY(:,i).^2);
    temperature(i) = (sys_mass/(2*K_B))*(vel(i));
end
% K_B * temp = (m/2)<(vx^2 + vy^2)>

% Plot results
% Press any key to move fowards 2 picoseconds
for i = 2:(T+2)
    plot(x(:,i), y(:,i), 'o');
    axis([0 2*sigma*L 0 2*sigma*L]);
    title(['Argon molecules at t = ' num2str((i-2)*2) ' picoseconds']);
    pause;
end
```